VISIT…

LANZAROTE
Caliente.COM

Computer Vision

A MODERN APPROACH

FORSYTH ▪ PONCE

CONTENTS

I IMAGE FORMATION 1

1 RADIOMETRY — MEASURING LIGHT 3
- 1.1 Light in Space 3
  - 1.1.1 Foreshortening 3
  - 1.1.2 Solid Angle 4
  - 1.1.3 Radiance 6
- 1.2 Light at Surfaces 8
  - 1.2.1 Simplifying Assumptions 9
  - 1.2.2 The Bidirectional Reflectance Distribution Function 9
- 1.3 Important Special Cases 11
  - 1.3.1 Radiosity 11
  - 1.3.2 Directional Hemispheric Reflectance 12
  - 1.3.3 Lambertian Surfaces and Albedo 12
  - 1.3.4 Specular Surfaces 13
  - 1.3.5 The Lambertian + Specular Model 14
- 1.4 Quick Reference: Radiometric Terminology for Light 16
- 1.5 Quick Reference: Radiometric Properties of Surfaces 17
- 1.6 Quick Reference: Important Types of Surface 18
- 1.7 Notes 19
- 1.8 Assignments 19

2 SOURCES, SHADOWS AND SHADING 21
- 2.1 Radiometric Properties of Light Sources 21
- 2.2 Qualitative Radiometry 22
- 2.3 Sources and their Effects 23
  - 2.3.1 Point Sources 24

2.3.2 Line Sources 26
2.3.3 Area Sources 27
2.4 Local Shading Models 28
2.4.1 Local Shading Models for Point Sources 28
2.4.2 Area Sources and their Shadows 31
2.4.3 Ambient Illumination 31
2.5 Application: Photometric Stereo 33
2.5.1 Normal and Albedo from Many Views 36
2.5.2 Shape from Normals 37
2.6 Interreflections: Global Shading Models 40
2.6.1 An Interreflection Model 42
2.6.2 Solving for Radiosity 43
2.6.3 The qualitative effects of interreflections 45
2.7 Notes 47
2.8 Assignments 50
2.8.1 Exercises 50
2.8.2 Programming Assignments 51

3 COLOUR 53
3.1 The Physics of Colour 53
3.1.1 Radiometry for Coloured Lights: Spectral Quantities 53
3.1.2 The Colour of Surfaces 54
3.1.3 The Colour of Sources 55
3.2 Human Colour Perception 58
3.2.1 Colour Matching 58
3.2.2 Colour Receptors 61
3.3 Representing Colour 63
3.3.1 Linear Colour Spaces 63
3.3.2 Non-linear Colour Spaces 68
3.3.3 Spatial and Temporal Effects 73
3.4 Application: Finding Specularities 73
3.5 Surface Colour from Image Colour 77
3.5.1 Surface Colour Perception in People 77
3.5.2 Inferring Lightness 80
3.5.3 A Model for Image Colour 83
3.5.4 Surface Colour from Finite Dimensional Linear Models 86
3.6 Notes 89

3.6.1 Trichromacy and Colour Spaces 89
3.6.2 Lightness and Colour Constancy 90
3.6.3 Colour in Recognition 91
3.7 Assignments 91

II IMAGE MODELS 94

4 GEOMETRIC IMAGE FEATURES 96
4.1 Elements of Differential Geometry 100
4.1.1 Curves 100
4.1.2 Surfaces 105
Application: The shape of specularities 109
4.2 Contour Geometry 112
4.2.1 The Occluding Contour and the Image Contour 113
4.2.2 The Cusps and Inflections of the Image Contour 114
4.2.3 Koenderink's Theorem 115
4.3 Notes 117
4.4 Assignments 118

5 ANALYTICAL IMAGE FEATURES 120
5.1 Elements of Analytical Euclidean Geometry 120
5.1.1 Coordinate Systems and Homogeneous Coordinates 121
5.1.2 Coordinate System Changes and Rigid Transformations 124
5.2 Geometric Camera Parameters 129
5.2.1 Intrinsic Parameters 129
5.2.2 Extrinsic Parameters 132
5.2.3 A Characterization of Perspective Projection Matrices 132
5.3 Calibration Methods 133
5.3.1 A Linear Approach to Camera Calibration 134
Technique: Linear Least Squares Methods 135
5.3.2 Taking Radial Distortion into Account 139
5.3.3 Using Straight Lines for Calibration 140
5.3.4 Analytical Photogrammetry 143
Technique: Non-Linear Least Squares Methods 145
5.4 Notes 147
5.5 Assignments 147

6 AN INTRODUCTION TO PROBABILITY 150
- 6.1 Probability in Discrete Spaces 151
  - 6.1.1 Probability: the P-function 151
  - 6.1.2 Conditional Probability 153
  - 6.1.3 Choosing P 153
- 6.2 Probability in Continuous Spaces 159
  - 6.2.1 Event Structures for Continuous Spaces 159
  - 6.2.2 Representing a P-function for the Real Line 160
  - 6.2.3 Probability Densities 161
- 6.3 Random Variables 161
  - 6.3.1 Conditional Probability and Independence 162
  - 6.3.2 Expectations 163
  - 6.3.3 Joint Distributions and Marginalization 165
- 6.4 Standard Distributions and Densities 165
  - 6.4.1 The Normal Distribution 167
- 6.5 Probabilistic Inference 167
  - 6.5.1 The Maximum Likelihood Principle 168
  - 6.5.2 Priors, Posteriors and Bayes' rule 170
  - 6.5.3 Bayesian Inference 170
  - 6.5.4 Open Issues 177
- 6.6 Discussion 178

III EARLY VISION: ONE IMAGE 180

7 LINEAR FILTERS 182
- 7.1 Linear Filters and Convolution 182
  - 7.1.1 Convolution 182
  - 7.1.2 Example: Smoothing by Averaging 183
  - 7.1.3 Example: Smoothing with a Gaussian 185
- 7.2 Shift invariant linear systems 186
  - 7.2.1 Discrete Convolution 188
  - 7.2.2 Continuous Convolution 190
  - 7.2.3 Edge Effects in Discrete Convolutions 192
- 7.3 Spatial Frequency and Fourier Transforms 193
  - 7.3.1 Fourier Transforms 193
- 7.4 Sampling and Aliasing 197
  - 7.4.1 Sampling 198

7.4.2 Aliasing 201
7.4.3 Smoothing and Resampling 202
7.5 Technique: Scale and Image Pyramids 204
7.5.1 The Gaussian Pyramid 205
7.5.2 Applications of Scaled Representations 206
7.5.3 Scale Space 208
7.6 Discussion 211
7.6.1 Real Imaging Systems vs Shift-Invariant Linear Systems 211
7.6.2 Scale 212

8 EDGE DETECTION 214
8.1 Estimating Derivatives with Finite Differences 214
8.1.1 Differentiation and Noise 216
8.1.2 Laplacians and edges 217
8.2 Noise 217
8.2.1 Additive Stationary Gaussian Noise 219
8.3 Edges and Gradient-based Edge Detectors 224
8.3.1 Estimating Gradients 224
8.3.2 Choosing a Smoothing Filter 225
8.3.3 Why Smooth with a Gaussian? 227
8.3.4 Derivative of Gaussian Filters 229
8.3.5 Identifying Edge Points from Filter Outputs 230
8.4 Commentary 234

9 FILTERS AND FEATURES 237
9.1 Filters as Templates 237
9.1.1 Convolution as a Dot Product 237
9.1.2 Changing Basis 238
9.2 Human Vision: Filters and Primate Early Vision 239
9.2.1 The Visual Pathway 239
9.2.2 How the Visual Pathway is Studied 241
9.2.3 The Response of Retinal Cells 241
9.2.4 The Lateral Geniculate Nucleus 242
9.2.5 The Visual Cortex 243
9.2.6 A Model of Early Spatial Vision 246
9.3 Technique: Normalised Correlation and Finding Patterns 248
9.3.1 Controlling the Television by Finding Hands by Normalised Correlation 248

9.4 Corners and Orientation Representations 249
9.5 Advanced Smoothing Strategies and Non-linear Filters 252
  9.5.1 More Noise Models 252
  9.5.2 Robust Estimates 253
  9.5.3 Median Filters 254
  9.5.4 Mathematical morphology: erosion and dilation 257
  9.5.5 Anisotropic Scaling 258
9.6 Commentary 259

10 TEXTURE 261
10.1 Representing Texture 263
  10.1.1 Extracting Image Structure with Filter Banks 263
10.2 Analysis (and Synthesis) Using Oriented Pyramids 268
  10.2.1 The Laplacian Pyramid 269
  10.2.2 Oriented Pyramids 272
10.3 Application: Synthesizing Textures for Rendering 272
  10.3.1 Homogeneity 274
  10.3.2 Synthesis by Matching Histograms of Filter Responses 275
  10.3.3 Synthesis by Sampling Conditional Densities of Filter Responses 280
  10.3.4 Synthesis by Sampling Local Models 284
10.4 Shape from Texture: Planes and Isotropy 286
  10.4.1 Recovering the Orientation of a Plane from an Isotropic Texture 288
  10.4.2 Recovering the Orientation of a Plane from an Homogeneity Assumption 290
  10.4.3 Shape from Texture for Curved Surfaces 291
10.5 Notes 292
  10.5.1 Shape from Texture 293

IV EARLY VISION: MULTIPLE IMAGES 295

11 THE GEOMETRY OF MULTIPLE VIEWS 297
11.1 Two Views 298
  11.1.1 Epipolar Geometry 298
  11.1.2 The Calibrated Case 299
  11.1.3 Small Motions 300
  11.1.4 The Uncalibrated Case 301
  11.1.5 Weak Calibration 302

11.2 Three Views 305
    11.2.1 Trifocal Geometry 307
    11.2.2 The Calibrated Case 307
    11.2.3 The Uncalibrated Case 309
    11.2.4 Estimation of the Trifocal Tensor 310
11.3 More Views 311
11.4 Notes 317
11.5 Assignments 319

12 STEREOPSIS 321
12.1 Reconstruction 323
    12.1.1 Camera Calibration 324
    12.1.2 Image Rectification 325
Human Vision: Stereopsis 327
12.2 Binocular Fusion 331
    12.2.1 Correlation 331
    12.2.2 Multi-Scale Edge Matching 333
    12.2.3 Dynamic Programming 336
12.3 Using More Cameras 338
    12.3.1 Trinocular Stereo 338
    12.3.2 Multiple-Baseline Stereo 340
12.4 Notes 341
12.5 Assignments 343

13 AFFINE STRUCTURE FROM MOTION 345
13.1 Elements of Affine Geometry 346
13.2 Affine Structure from Two Images 349
    13.2.1 The Affine Structure-from-Motion Theorem 350
    13.2.2 Rigidity and Metric Constraints 351
13.3 Affine Structure from Multiple Images 351
    13.3.1 The Affine Structure of Affine Image Sequences 352
Technique: Singular Value Decomposition 353
    13.3.2 A Factorization Approach to Affine Motion Analysis 353
13.4 From Affine to Euclidean Images 356
    13.4.1 Euclidean Projection Models 357
    13.4.2 From Affine to Euclidean Motion 358
13.5 Affine Motion Segmentation 360
    13.5.1 The Reduced Echelon Form of the Data Matrix 360

13.5.2 The Shape Interaction Matrix 360
13.6 Notes 362
13.7 Assignments 363

14 PROJECTIVE STRUCTURE FROM MOTION 365
14.1 Elements of Projective Geometry 366
14.1.1 Projective Bases and Projective Coordinates 366
14.1.2 Projective Transformations 368
14.1.3 Affine and Projective Spaces 370
14.1.4 Hyperplanes and Duality 371
14.1.5 Cross-Ratios 372
14.1.6 Application: Parameterizing the Fundamental Matrix 375
14.2 Projective Scene Reconstruction from Two Views 376
14.2.1 Analytical Scene Reconstruction 376
14.2.2 Geometric Scene Reconstruction 378
14.3 Motion Estimation from Two or Three Views 379
14.3.1 Motion Estimation from Fundamental Matrices 379
14.3.2 Motion Estimation from Trifocal Tensors 381
14.4 Motion Estimation from Multiple Views 382
14.4.1 A Factorization Approach to Projective Motion Analysis 383
14.4.2 Bundle Adjustment 386
14.5 From Projective to Euclidean Structure and Motion 386
14.5.1 Metric Upgrades from (Partial) Camera Calibration 387
14.5.2 Metric Upgrades from Minimal Assumptions 389
14.6 Notes 392
14.7 Assignments 394

V MID-LEVEL VISION 399

15 SEGMENTATION USING CLUSTERING METHODS 401
15.1 Human vision: Grouping and Gestalt 403
15.2 Applications: Shot Boundary Detection, Background Subtraction and Skin Finding 407
15.2.1 Background Subtraction 407
15.2.2 Shot Boundary Detection 408
15.2.3 Finding Skin Using Image Colour 410
15.3 Image Segmentation by Clustering Pixels 411

15.3.1 Simple Clustering Methods 411
15.3.2 Segmentation Using Simple Clustering Methods 413
15.3.3 Clustering and Segmentation by K-means 415
15.4 Segmentation by Graph-Theoretic Clustering 417
15.4.1 Basic Graphs 418
15.4.2 The Overall Approach 420
15.4.3 Affinity Measures 420
15.4.4 Eigenvectors and Segmentation 424
15.4.5 Normalised Cuts 427
15.5 Discussion 430

16 FITTING 436
16.1 The Hough Transform 437
16.1.1 Fitting Lines with the Hough Transform 437
16.1.2 Practical Problems with the Hough Transform 438
16.2 Fitting Lines 440
16.2.1 Least Squares, Maximum Likelihood and Parameter Estimation 441
16.2.2 Which Point is on Which Line? 444
16.3 Fitting Curves 445
16.3.1 Implicit Curves 446
16.3.2 Parametric Curves 449
16.4 Fitting to the Outlines of Surfaces 450
16.4.1 Some Relations Between Surfaces and Outlines 451
16.4.2 Clustering to Form Symmetries 453
16.5 Discussion 457

17 SEGMENTATION AND FITTING USING PROBABILISTIC METHODS 460
17.1 Missing Data Problems, Fitting and Segmentation 461
17.1.1 Missing Data Problems 461
17.1.2 The EM Algorithm 463
17.1.3 Colour and Texture Segmentation with EM 469
17.1.4 Motion Segmentation and EM 470
17.1.5 The Number of Components 474
17.1.6 How Many Lines are There? 474
17.2 Robustness 475
17.2.1 Explicit Outliers 475
17.2.2 M-estimators 477

17.2.3 RANSAC 480
17.3 How Many are There? 483
17.3.1 Basic Ideas 484
17.3.2 AIC — An Information Criterion 484
17.3.3 Bayesian methods and Schwartz' BIC 485
17.3.4 Description Length 486
17.3.5 Other Methods for Estimating Deviance 486
17.4 Discussion 487

18 TRACKING 489
18.1 Tracking as an Abstract Inference Problem 490
18.1.1 Independence Assumptions 490
18.1.2 Tracking as Inference 491
18.1.3 Overview 492
18.2 Linear Dynamic Models and the Kalman Filter 492
18.2.1 Linear Dynamic Models 492
18.2.2 Kalman Filtering 497
18.2.3 The Kalman Filter for a 1D State Vector 497
18.2.4 The Kalman Update Equations for a General State Vector 499
18.2.5 Forward-Backward Smoothing 500
18.3 Non-Linear Dynamic Models 505
18.3.1 Unpleasant Properties of Non-Linear Dynamics 508
18.3.2 Difficulties with Likelihoods 509
18.4 Particle Filtering 511
18.4.1 Sampled Representations of Probability Distributions 511
18.4.2 The Simplest Particle Filter 515
18.4.3 A Workable Particle Filter 518
18.4.4 If's, And's and But's — Practical Issues in Building Particle Filters 519
18.5 Data Association 523
18.5.1 Choosing the Nearest — Global Nearest Neighbours 523
18.5.2 Gating and Probabilistic Data Association 524
18.6 Applications and Examples 527
18.6.1 Vehicle Tracking 528
18.6.2 Finding and Tracking People 532
18.7 Discussion 538
II Appendix: The Extended Kalman Filter, or EKF 540

VI HIGH-LEVEL VISION 542

19 CORRESPONDENCE AND POSE CONSISTENCY 544

- 19.1 Initial Assumptions 544
  - 19.1.1 Obtaining Hypotheses 545
- 19.2 Obtaining Hypotheses by Pose Consistency 546
  - 19.2.1 Pose Consistency for Perspective Cameras 547
  - 19.2.2 Affine and Projective Camera Models 549
  - 19.2.3 Linear Combinations of Models 551
- 19.3 Obtaining Hypotheses by Pose Clustering 553
- 19.4 Obtaining Hypotheses Using Invariants 554
  - 19.4.1 Invariants for Plane Figures 554
  - 19.4.2 Geometric Hashing 559
  - 19.4.3 Invariants and Indexing 560
- 19.5 Verification 564
  - 19.5.1 Edge Proximity 565
  - 19.5.2 Similarity in Texture, Pattern and Intensity 567
  - 19.5.3 Example: Bayes Factors and Verification 567
- 19.6 Application: Registration in Medical Imaging Systems 568
  - 19.6.1 Imaging Modes 569
  - 19.6.2 Applications of Registration 570
  - 19.6.3 Geometric Hashing Techniques in Medical Imaging 571
- 19.7 Curved Surfaces and Alignment 573
- 19.8 Discussion 576

20 FINDING TEMPLATES USING CLASSIFIERS 581

- 20.1 Classifiers 582
  - 20.1.1 Using Loss to Determine Decisions 582
  - 20.1.2 Overview: Methods for Building Classifiers 584
  - 20.1.3 Example: A Plug-in Classifier for Normal Class-conditional Densities 586
  - 20.1.4 Example: A Non-Parametric Classifier using Nearest Neighbours 587
  - 20.1.5 Estimating and Improving Performance 588
- 20.2 Building Classifiers from Class Histograms 590
  - 20.2.1 Finding Skin Pixels using a Classifier 591
  - 20.2.2 Face Finding Assuming Independent Template Responses 592
- 20.3 Feature Selection 595

20.3.1 Principal Component Analysis 595
20.3.2 Canonical Variates 597
20.4 Neural Networks 601
20.4.1 Key Ideas 601
20.4.2 Minimizing the Error 606
20.4.3 When to Stop Training 610
20.4.4 Finding Faces using Neural Networks 610
20.4.5 Convolutional Neural Nets 612
20.5 The Support Vector Machine 615
20.5.1 Support Vector Machines for Linearly Separable Datasets 616
20.5.2 Finding Pedestrians using Support Vector Machines 618
20.6 Conclusions 622
II Appendix: Support Vector Machines for Datasets that are not Linearly Separable 624
III Appendix: Using Support Vector Machines with Non-Linear Kernels 625

21 RECOGNITION BY RELATIONS BETWEEN TEMPLATES 627
21.1 Finding Objects by Voting on Relations between Templates 628
21.1.1 Describing Image Patches 628
21.1.2 Voting and a Simple Generative Model 629
21.1.3 Probabilistic Models for Voting 630
21.1.4 Voting on Relations 632
21.1.5 Voting and 3D Objects 632
21.2 Relational Reasoning using Probabilistic Models and Search 633
21.2.1 Correspondence and Search 633
21.2.2 Example: Finding Faces 636
21.3 Using Classifiers to Prune Search 639
21.3.1 Identifying Acceptable Assemblies Using Projected Classifiers 640
21.3.2 Example: Finding People and Horses Using Spatial Relations 640
21.4 Technique: Hidden Markov Models 643
21.4.1 Formal Matters 644
21.4.2 Computing with Hidden Markov Models 645
21.4.3 Varieties of HMM's 652
21.5 Application: Hidden Markov Models and Sign Language Understanding 654
21.6 Application: Finding People with Hidden Markov Models 659
21.7 Frames and Probability Models 662
21.7.1 Representing Coordinate Frames Explicitly in a Probability Model 664

21.7.2 Using a Probability Model to Predict Feature Positions 666
21.7.3 Building Probability Models that are Frame-Invariant 668
21.7.4 Example: Finding Faces Using Frame Invariance 669
21.8 Conclusions 669

22 ASPECT GRAPHS 672
22.1 Differential Geometry and Visual Events 677
22.1.1 The Geometry of the Gauss Map 677
22.1.2 Asymptotic Curves 679
22.1.3 The Asymptotic Spherical Map 681
22.1.4 Local Visual Events 682
22.1.5 The Bitangent Ray Manifold 684
22.1.6 Multilocal Visual Events 686
22.1.7 Remarks 687
22.2 Computing the Aspect Graph 689
22.2.1 Step 1: Tracing Visual Events 690
22.2.2 Step 2: Constructing the Regions 691
22.2.3 Remaining Steps of the Algorithm 692
22.2.4 An Example 692
22.3 Aspect Graphs and Object Recognition 696
22.4 Notes 696
22.5 Assignments 697

VII APPLICATIONS AND TOPICS 699

23 RANGE DATA 701
23.1 Active Range Sensors 701
23.2 Range Data Segmentation 704
Technique: Analytical Differential Geometry 705
23.2.1 Finding Step and Roof Edges in Range Images 707
23.2.2 Segmenting Range Images into Planar Regions 712
23.3 Range Image Registration and Model Construction 714
Technique: Quaternions 715
23.3.1 Registering Range Images Using the Iterative Closest-Point Method 716
23.3.2 Fusing Multiple Range Images 719
23.4 Object Recognition 720

    23.4.1 Matching Piecewise-Planar Surfaces Using Interpretation Trees 721
    23.4.2 Matching Free-Form Surfaces Using Spin Images 724
  23.5 Notes 729
  23.6 Assignments 730

24 APPLICATION: FINDING IN DIGITAL LIBRARIES 732
  24.1 Background 733
    24.1.1 What do users want? 733
    24.1.2 What can tools do? 735
  24.2 Appearance 736
    24.2.1 Histograms and correlograms 737
    24.2.2 Textures and textures of textures 738
  24.3 Finding 745
    24.3.1 Annotation and segmentation 748
    24.3.2 Template matching 749
    24.3.3 Shape and correspondence 751
  24.4 Video 754
  24.5 Discussion 756

25 APPLICATION: IMAGE-BASED RENDERING 758
  25.1 Constructing 3D Models from Image Sequences 759
    25.1.1 Scene Modeling from Registered Images 759
    25.1.2 Scene Modeling from Unregistered Images 767
  25.2 Transfer-Based Approaches to Image-Based Rendering 771
    25.2.1 Affine View Synthesis 772
    25.2.2 Euclidean View Synthesis 775
  25.3 The Light Field 778
  25.4 Notes 782
  25.5 Assignments 784

Part I

IMAGE FORMATION

Chapter 1

RADIOMETRY — MEASURING LIGHT

In this chapter, we introduce a vocabulary with which we can describe the behaviour of light. There are no vision algorithms, but definitions and ideas that will be useful later on. Some readers may find more detail here than they really want; for their benefit, sections 1.4, 1.5 and 1.6 give quick definitions of the main terms we use later on.

1.1 Light in Space

The measurement of light is a field in itself, known as **radiometry**. We need a series of units that describe how energy is transferred from light sources to surface patches, and what happens to the energy when it arrives at a surface. The first matter to study is the behaviour of light in space.

1.1.1 Foreshortening

At each point on a piece of surface is a hemisphere of directions, along which light can arrive or leave (figure 1.1). Two sources that generate the same pattern on this input hemisphere must have the same effect on the surface at this point (because an observer at the surface can't tell them apart). This applies to sources, too; two surfaces that generate the same pattern on a source's output hemisphere must receive the same amount of energy from the source.

This means that the orientation of the surface patch with respect to the direction in which the illumination is travelling is important. As a source is tilted with respect to the direction in which the illumination is travelling, it "looks smaller" to a patch of surface. Similarly, as a patch is tilted with respect to the direction in which the illumination is travelling, it "looks smaller" to the source.

The effect is known as **foreshortening**. Foreshortening is important, because *from the point of view of the source* a small patch appears the same as a large patch that is heavily foreshortened, and so must receive the same energy.

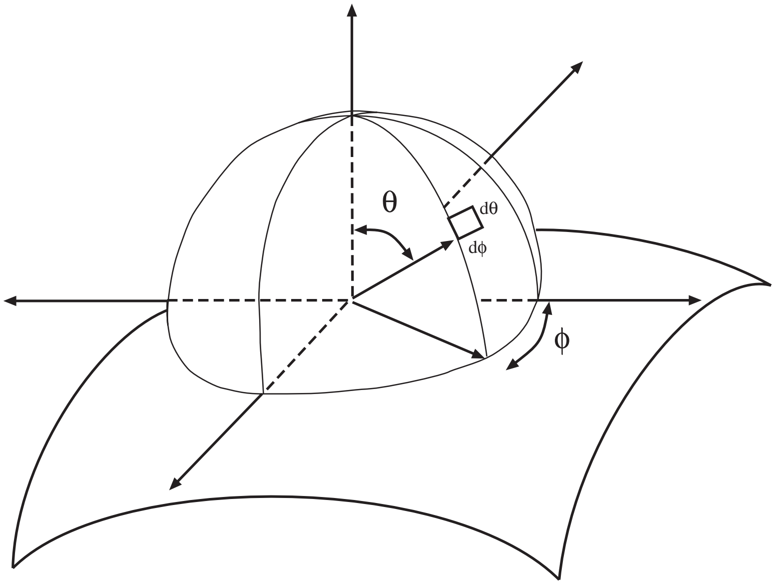


Figure 1.1. A point on a surface sees the world along a hemisphere of directions centered at the point; the surface normal is used to orient the hemisphere, to obtain the θ, ϕ coordinate system that we use consistently from now on to describe angular coordinates on this hemisphere. Usually in radiation problems we compute the brightness of the surface by summing effects due to all incoming directions, so that the fact we have given no clear way to determine the direction in which $\phi = 0$ is not a problem.

1.1.2 Solid Angle

The pattern a source generates on an input hemisphere can be described by the **solid angle** that the source subtends. Solid angle is defined by analogy with angle on the plane.

The angle subtended on the plane by an infinitesimal line segment of length dl at a point $\boldsymbol{p}$ can be obtained by projecting the line segment onto the unit circle whose center is at $\boldsymbol{p}$; the length of the result is the required angle in radians (see Figure 1.2). Because the line segment is infinitesimally short, it subtends an infinitesimally small angle which depends on the distance to the center of the circle and on the orientation of the line:

$$d\phi = \frac{dl \cos\theta}{r}$$

and the angle subtended by a curve can be obtained by breaking it into infinitesimal segments and summing (integration!).

Similarly, the solid angle subtended by a patch of surface at a point $\boldsymbol{x}$ is obtained by projecting the patch onto the unit sphere whose center is at $\boldsymbol{x}$; the area of the result is the required solid angle, whose unit is now **steradians**. Solid angle is usually denoted by the symbol ω. Notice that solid angle captures the intuition in foreshortening — patches that "look the same" on the input hemisphere subtend the same solid angle.

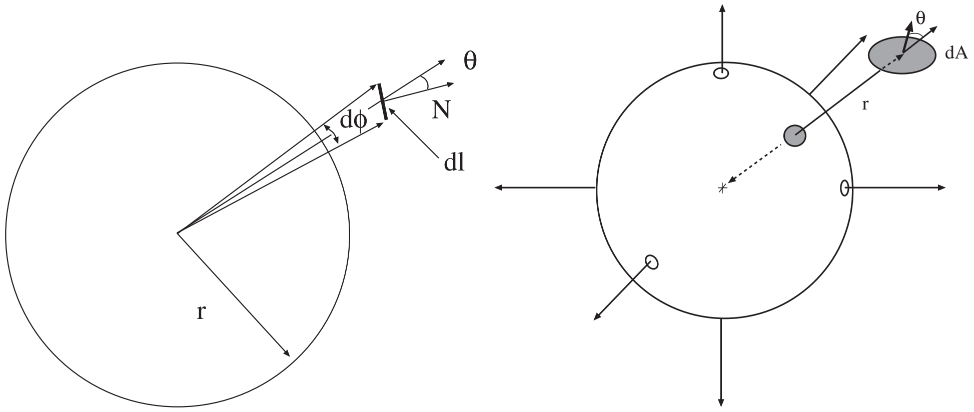


Figure 1.2. **Top:** The angle subtended by a curve segment at a particular point is obtained by projecting the curve onto the unit circle whose center is at that point, and then measuring the length of the projection. For a small segment, the angle is $(1/r)dl\cos\theta$. **Bottom:** A sphere, illustrating the concept of solid angle. The small circles surrounding the coordinate axes are to help you see the drawing as a 3D surface. An infinitesimal patch of surface is projected onto the unit sphere centered at the relevant point; the resulting area is the solid angle of the patch. In this case, the patch is small, so that the angle is $(1/r^2)dA\cos\theta$.

If the area of the patch dA is small (as suggested by the infinitesimal form), then the infinitesimal solid angle it subtends is easily computed in terms of the area of the patch and the distance to it as

$$d\omega = \frac{dA\cos\theta}{r^2}$$

where the terminology is given in Figure 1.2.

Solid angle can be written in terms of the usual angular coordinates on a sphere (illustrated in Figure 1.2). From figure 1.1 and the expression for the length of circular arcs, we have that infinitesimal steps ($d\theta$, $d\phi$) in the angles θ and ϕ cut out a region of solid angle on a sphere given by:

$$d\omega = \sin\theta d\theta d\phi$$

Both of these expressions are worth remembering, as they turn out to be useful for a variety of applications.

1.1.3 Radiance

The distribution of light in space is a function of position and direction. For example, consider shining a torch with a narrow beam in an empty room at night — we need to know where the torch is shining from, and in what direction it is shining. The effect of the illumination can be represented in terms of the power an infinitesimal patch of surface would receive if it were inserted into space at a particular point and orientation. We will use this approach to obtain a unit of measurement.

Definition of Radiance

The appropriate unit for measuring the distribution of light in space is **radiance**, which is defined as:

> the amount of energy travelling at some point in a specified direction, per unit time, per unit area *perpendicular to the direction of travel*, per unit solid angle (from [**?**])

The units of radiance are watts per square meter per steradian ($Wm^{-2}sr^{-1}$). It is important to remember that the square meters in these units are *foreshortened*, i.e. perpendicular to the direction of travel. This means that a small patch viewing a source frontally collects more energy than the same patch viewing a source radiance along a nearly tangent direction — the amount of energy a patch collects from a source depends both on how large the source looks from the patch *and* on how large the patch looks from the source.

Radiance is a function of position and direction (the torch with a narrow beam is a good model to keep in mind — you can move the torch around, and point the beam in different directions). The radiance at a point in space is usually denoted $L(\boldsymbol{x}, \text{direction})$, where $\boldsymbol{x}$ is a coordinate for position — which can be a point in free space or a point on a surface — and we use some mechanism for specifying direction.

One way to specify direction is to use (θ, ϕ) coordinates established using some surface normal. Another is to write $\boldsymbol{x}_1 \to \boldsymbol{x}_2$, meaning the direction from point $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$. We shall use both, depending on which is convenient for the problem at hand.

Radiance is Constant Along a Straight Line

For the vast majority of important vision problems, it is safe to assume that light does not interact with the medium through which it travels — i.e. that we are in a vacuum. Radiance has the highly desirable property that, for two points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ (which have a line of sight between them), the radiance leaving $\boldsymbol{p}_1$ in the direction of $\boldsymbol{p}_2$ is the same as the radiance arriving at $\boldsymbol{p}_2$ from the direction of $\boldsymbol{p}_1$.

The following proof may look vacuous at first glance; it's worth studying carefully, because it is the key to a number of other computations. Figure 1.3 shows a patch of surface radiating in a particular direction. From the definition, if the

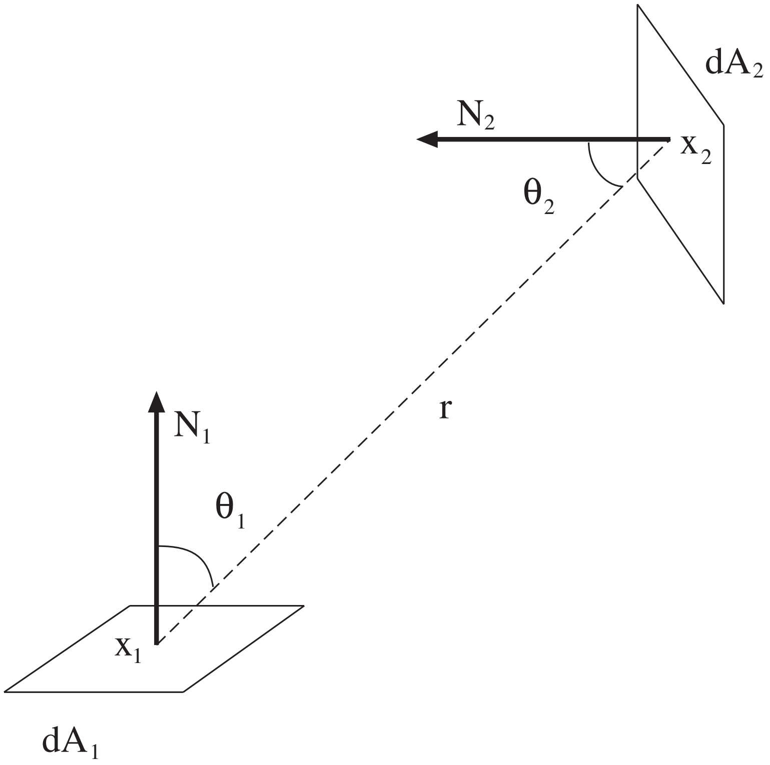


Figure 1.3. Light intensity is best measured in radiance, because radiance does not go down along straight line paths in a vacuum (or, for reasonable distances, in clear air). This is shown by an energy conservation argument in the text, where one computes the energy transferred from a patch dA_1 to a patch dA_2

radiance at the patch is $L(\boldsymbol{x}_1, \theta, \phi)$, then the energy transmitted by the patch into an infinitesimal region of solid angle $d\omega$ around the direction θ, ϕ in time dt is

$$L(\boldsymbol{x}_1, \theta, \phi)(\cos\theta_1 dA_1)(d\omega)(dt),$$

(i.e. radiance times the foreshortened area of the patch times the solid angle into which the power is radiated times the time for which the power is radiating).

Now consider two patches, one at $\boldsymbol{x}_1$ with area dA_1 and the other at $\boldsymbol{x}_2$ with area dA_2 (see Figure 1.3). To avoid confusion with angular coordinate systems, write the angular direction from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ as $\boldsymbol{x}_1 \to \boldsymbol{x}_2$. The angles θ_1 and θ_2 are as defined in figure 1.3.

The radiance leaving $\boldsymbol{x}_1$ in the direction of $\boldsymbol{x}_2$ is $L(\boldsymbol{x}_1, \boldsymbol{x}_1 \to \boldsymbol{x}_2)$ and the radiance arriving at $\boldsymbol{x}_2$ from the direction of $\boldsymbol{x}_1$ is $L(\boldsymbol{x}_2, \boldsymbol{x}_1 \to \boldsymbol{x}_2)$.

This means that, in time dt, the energy leaving $\boldsymbol{x}_1$ towards $\boldsymbol{x}_2$ is

$$d^3E_{1\to 2} = L(\boldsymbol{x}_1, \boldsymbol{x}_1 \to \boldsymbol{x}_2)\cos\theta_1 d\omega_{2(1)} dA_1 dt$$

where $d\omega_{2(1)}$ is the solid angle subtended by patch 2 at patch 1 (energy emitted into this solid angle arrives at 2; all the rest disappears into the void). The notation $d^3E_{1\to 2}$ implies that there are three infinitesimal terms involved.

From the expression for solid angle above,

$$d\omega_{2(1)} = \frac{\cos\theta_2 dA_2}{r^2}$$

Now the energy leaving 1 for 2 is:

$$\begin{aligned} d^3E_{1\to 2} &= L(\boldsymbol{x}_1, \boldsymbol{x}_1 \to \boldsymbol{x}_2)\cos\theta_1 d\omega_{2(1)} dA_1 dt \\ &= L(\boldsymbol{x}_1, \boldsymbol{x}_1 \to \boldsymbol{x}_2)\frac{\cos\theta_1 \cos\theta_2 dA_2 dA_1 dt}{r^2} \end{aligned}$$

Because the medium is a vacuum, it does not absorb energy, so that the energy arriving at 2 from 1 is the same as the energy leaving 1 in the direction of 2. The energy arriving at 2 from 1 is:

$$\begin{aligned} d^3E_{1\to 2} &= L(\boldsymbol{x}_2, \boldsymbol{x}_1 \to \boldsymbol{x}_2)\cos\theta_2 d\omega_{1(2)} dA_2 dt \\ &= L(\boldsymbol{x}_2, \boldsymbol{x}_1 \to \boldsymbol{x}_2)\frac{\cos\theta_2 \cos\theta_1 dA_1 dA_2 dt}{r^2} \end{aligned}$$

which means that $L(\boldsymbol{x}_2, \boldsymbol{x}_1 \to \boldsymbol{x}_2) = L(\boldsymbol{x}_1, \theta, \phi)$, so that *radiance is constant along (unoccluded) straight lines.*

1.2 Light at Surfaces

When light strikes a surface, it may be absorbed, transmitted, or scattered; usually, a combination of these effects occur. For example, light arriving at skin can be scattered at various depths into tissue and reflected from blood or from melanin in there; can be absorbed; or can be scattered tangential to the skin within a film of oil and then escape at some distant point.

The picture is complicated further by the willingness of some surfaces to absorb light at one wavelength, and then radiate light at a different wavelength as a result. This effect, known as **fluorescence**, is fairly common: scorpions fluoresce visible light under x-ray illumination; human teeth fluoresce faint blue under ultraviolet light (nylon underwear tends to fluoresce, too, and false teeth generally do not — the resulting embarrassments led to the demise of uv lights in discotheques); and laundry can be made to look bright by washing powders that fluoresce under ultraviolet light. Furthermore, a surface that is warm enough emits light in the visible range.

1.2.1 Simplifying Assumptions

It is common to assume that all effects are local, and can be explained with a macroscopic model with no fluorescence or emission. This is a reasonable model for the kind of surfaces and decisions that are common in vision. In this model:

- the radiance leaving a point on a surface is due only to radiance arriving at this point (although radiance may change *directions* at a point on a surface, we assume that it does not skip from point to point);

- we assume that all light leaving a surface at a given wavelength is due to light arriving at that wavelength;

- we assume that the surfaces do not generate light internally, and treat sources separately.

1.2.2 The Bidirectional Reflectance Distribution Function

We wish to describe the relationship between incoming illumination and reflected light. This will be a function of both the direction in which light arrives at a surface and the direction in which it leaves.

Irradiance

The appropriate unit for representing incoming power which is **irradiance**, defined as:

> incident power per unit area *not foreshortened*.

A surface illuminated by radiance $L_i(\boldsymbol{x}, \theta_i, \phi_i)$ coming in from a differential region of solid angle $d\omega$ at angles (θ_i, ϕ_i) receives irradiance

$$L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos \theta_i d\omega$$

where we have multiplied the radiance by the foreshortening factor and by the solid angle to get irradiance. The main feature of this unit is that we could compute all the power incident on a surface at a point by summing the irradiance over the whole input hemisphere — which makes it the natural unit for *incoming* power.

The BRDF

The most general model of local reflection is the **bidirectional reflectance distribution function**, usually abbreviated BRDF. The BRDF is defined as

> the ratio of the radiance in the outgoing direction to the incident irradiance (after [**?**])

so that, if the surface of the preceding paragraph was to emit radiance $L_o(\boldsymbol{x}, \theta_o, \phi_o)$, its BRDF would be:

$$\rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) = \frac{L_o(\boldsymbol{x}, \theta_o, \phi_o)}{L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos \theta_i d\omega}$$

The BRDF has units of inverse steradians (sr^{-1}), and could vary from 0 (no light reflected in that direction) to infinity (unit radiance in an exit direction resulting from arbitrary small radiance in the incoming direction). The BRDF is symmetric in the incoming and outgoing direction, a fact known as the Helmholtz reciprocity principle.

Properties of the BRDF

The radiance leaving a surface due to irradiance *in a particular direction* is easily obtained from the definition of the BRDF:

$$L_o(\boldsymbol{x}, \theta_o, \phi_o) = \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos \theta_i d\omega$$

More interesting is the radiance leaving a surface due to its irradiance (whatever the direction of irradiance). We obtain this by summing over contributions from all incoming directions:

$$L_o(\boldsymbol{x}, \theta_o, \phi_o) = \int_\Omega \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos \theta_i d\omega$$

where Ω is the incoming hemisphere. From this we obtain the fact that the BRDF is not an arbitrary symmetric function in four variables.

To see this, assume that a surface is subjected to a radiance of $1/\cos\theta_i \; Wm^{-2}sr^{-1}$. This means that the total energy arriving at the surface is:

$$\begin{aligned} \int_\Omega \frac{1}{\cos\theta} \cos\theta d\omega &= \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \sin\theta d\theta d\phi \\ &= 2\pi \end{aligned}$$

We have assumed that any energy leaving at the surface leaves from the same point at which it arrived, and that no energy is generated within the surface. This means that the total energy leaving the surface must be less than or equal to the amount arriving. So we have

$$\begin{aligned} 2\pi &\geq \int_{\Omega_o} L_o(\boldsymbol{x}, \theta_o, \phi_o) \cos\theta_o d\omega_o \\ &= \int_{\Omega_o} \int_{\Omega_i} \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos\theta_i d\omega_i d\omega_o \\ &= \int_{\Omega_o} \int_{\Omega_i} \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) d\omega_i d\omega_o \\ &= \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) \sin\theta_i d\theta_i d\phi_i \sin\theta_o d\theta_o d\phi_o \end{aligned}$$

What this tells us is that, although the BRDF can be large for some pairs of incoming and outgoing angles, it can't be large for many.

1.3 Important Special Cases

Radiance is a fairly subtle quantity, because it depends on angle. This generality is sometimes essential — for example, for describing the distribution of light in space in the torch beam example above. As another example, fix a compact disc and illuminate its underside with a torch beam. The intensity and colour of light reflected from the surface depends very strongly on the angle from which the surface is viewed and on the angle from which it is illuminated. The CD example is worth trying, because it illustrates how strange the behaviour of reflecting surfaces can be; it also illustrates how accustomed we are to dealing with surfaces that do not behave in this way. For many surfaces — cotton cloth is one good example — the dependency of reflected light on angle is weak or non-existent, so that a system of units that are independent of angle is useful.

1.3.1 Radiosity

If the radiance leaving a surface is independent of exit angle, there is no point in describing it using a unit that explicitly depends on direction. The appropriate unit is **radiosity**, defined as

> the total power leaving a point on a surface per unit area on the surface (from [**?**])

Radiosity, which is usually written as $B(\boldsymbol{x})$ has units watts per square meter (Wm^{-2}). To obtain the radiosity of a surface at a point, we can sum the radiance leaving the surface at that point over the whole exit hemisphere. Thus, if $\boldsymbol{x}$ is a point on a surface emitting radiance $L(\boldsymbol{x}, \theta, \phi)$, the radiosity at that point will be:

$$B(\boldsymbol{x}) = \int_\Omega L(\boldsymbol{x}, \theta, \phi) \cos\theta d\omega$$

where Ω is the exit hemisphere and the term $\cos\theta$ turns foreshortened area into area (look at the definitions again!); $d\omega$ can be written in terms of θ, ϕ as above.

The Radiosity of a Surface with Constant Radiance

One result to remember is the relationship between the radiosity and the radiance of a surface patch *where the radiance is independent of angle.* In this case $L_o(x, \theta_o, \phi_o) = L_o(\boldsymbol{x})$. Now the radiosity can be obtained by summing the radiance leaving the surface over all the directions in which it leaves:

$$B(\boldsymbol{x}) \;=\; \int_\Omega L_o(\boldsymbol{x}) \cos\theta d\omega$$

$$
\begin{aligned}
&= L_o(\boldsymbol{x}) \int_0^{\frac{\pi}{2}} \int_0^{2\pi} \cos\theta \sin\theta d\phi d\theta \\
&= \pi L_o(\boldsymbol{x})
\end{aligned}
$$

1.3.2 Directional Hemispheric Reflectance

The BRDF is also a subtle quantity, and BRDF measurements are typically difficult, expensive and not particularly repeatable. This is because surface dirt and aging processes can have significant effects on BRDF measurements; for example, touching a surface will transfer oil to it, typically in little ridges (from the fingertips) which can act as lenses and make significant changes in the directional behaviour of the surface.

The light leaving many surfaces is largely independent of the exit angle. A natural measure of a surface's reflective properties in this case is the **directional-hemispheric reflectance**, usually termed ρ_{dh}, defined as:

> the fraction of the incident irradiance in a given direction that is reflected by the surface, whatever the direction of reflection (after [**?**])

The directional hemispheric reflectance of a surface is obtained by summing the radiance leaving the surface over all directions, and dividing by the irradiance in the direction of illumination, which gives:

$$
\begin{aligned}
\rho_{dh}(\theta_i, \phi_i) &= \frac{\int_\Omega L_o(\boldsymbol{x}, \theta_o, \phi_o) \cos\theta_o d\omega_o}{L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos\theta_i d\omega_i} \\
&= \int_\Omega \left\{ \frac{L_o(\boldsymbol{x}, \theta_o, \phi_o) \cos\theta_o}{L_i(\boldsymbol{x}, \theta_i, \phi_i) \cos\theta_i d\omega_i} \right\} d\omega_o \\
&= \int_\Omega \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) \cos\theta_o d\omega_o
\end{aligned}
$$

This property is dimensionless, and its value will lie between 0 and 1.

Directional hemispheric reflectance can be computed for any surface. For some surfaces, it will vary sharply with the direction of illumination. A good example is a surface with fine, symmetric triangular grooves which are black on one face and white on the other. If these grooves are sufficiently fine, it is reasonable to use a macroscopic description of the surface as flat, and with a directional hemispheric reflectance that is large along a direction pointing towards the white faces and small along that pointing towards the black.

1.3.3 Lambertian Surfaces and Albedo

For some surfaces the directional hemispheric reflectance does not depend on illumination direction. Examples of such surfaces include cotton cloth, many carpets, matte paper and matte paints. A formal model is given by a surface whose BRDF is independent of outgoing direction (and, by the reciprocity principle, of incoming direction as well). This means the radiance leaving the surface is independent of angle. Such surfaces are known as **ideal diffuse surfaces** or **Lambertian surfaces** (after George Lambert, who first formalised the idea).

It is natural to use radiosity as a unit to describe the energy leaving a Lambertian surface. For Lambertian surfaces, the directional hemispheric reflectance is independent of direction. In this case the directional hemispheric reflectance is often called their **diffuse reflectance** or **albedo** and written ρ_d. For a Lambertian surface with BRDF $\rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) = \rho$, we have:

$$\begin{aligned}
\rho_d &= \int_\Omega \rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) \cos\theta_o d\omega_o \\
&= \int_\Omega \rho \cos\theta_o d\omega_o \\
&= \rho \int_0^{\frac{\pi}{2}} \int_0^{2\pi} \cos\theta_o \sin\theta_o d\theta_o d\phi_o \\
&= \pi\rho
\end{aligned}$$

This fact is more often used in the form

$$\rho_{brdf} = \frac{\rho_d}{\pi}$$

a fact that is useful, and well worth remembering.

Because our sensations of brightness correspond (roughly!) to measurements of radiance, a Lambertian surface will look equally bright from any direction, whatever the direction along which it is illuminated. This gives a rough test for when a Lambertian approximation is appropriate.

1.3.4 Specular Surfaces

A second important class of surfaces are the glossy or mirror-like surfaces, often known as **specular** surfaces (after the Latin word *speculum*, a mirror). An ideal specular reflector behaves like an ideal mirror. Radiation arriving along a particular direction can leave only along the **specular direction**, obtained by reflecting the direction of incoming radiation about the surface normal. Usually some fraction of incoming radiation is absorbed; on an ideal specular surface, the same fraction of incoming radiation is absorbed for every direction, the rest leaving along the specular direction. The BRDF for an ideal specular surface has a curious form (exercise **??**), because radiation arriving in a particular direction can leave in only one direction.

Specular Lobes

Relatively few surfaces can be approximated as ideal specular reflectors. A fair test of whether a flat surface can be approximated as an ideal specular reflector is whether one could safely use it as a mirror. Good mirrors are suprisingly hard to make; up until recently, mirrors were made of polished metal. Typically, unless the

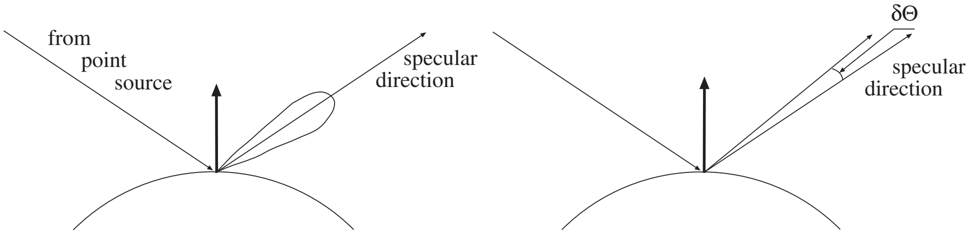


Figure 1.4. Specular surfaces commonly reflect light into a lobe of directions around the specular direction, where the intensity of the reflection depends on the direction, as shown on the left. Phong's model is used to describe the shape of this lobe, in terms of the offset angle from the specular direction.

metal is extremely highly polished and carefully maintained, radiation arriving in one direction leaves in a small lobe of directions around the specular direction. This results in a typical blurring effect. A good example is the bottom of a flat metal pie dish. If the dish is reasonably new, one can see a distorted image of one's face in the surface but it would be difficult to use as a mirror; a more battered dish reflects a selection of distorted blobs.

Larger specular lobes mean that the specular image is more heavily distorted and is darker (because the incoming radiance must be shared over a larger range of outgoing directions). Quite commonly it is possible to see only a specular reflection of relatively bright objects, like sources. Thus, in shiny paint or plastic surfaces, one sees a bright blob — often called a **specularity** — along the specular direction from light sources, but few other specular effects. It is not often necessary to model the shape of the specular lobe. When the shape of the lobe is modelled, the most common model is the **Phong** model, which assumes that only point light sources are specularly reflected. In this model, the radiance leaving a specular surface is proportional to $\cos^n(\delta\theta) = \cos^n(\theta_o - \theta_s)$, where θ_o is the exit angle, θ_s is the specular direction and n is a parameter. Large values of n lead to a narrow lobe and small, sharp specularities and small values lead to a broad lobe and large specularities with rather fuzzy boundaries.

1.3.5 The Lambertian + Specular Model

Relatively few surfaces are either ideal diffuse or perfectly specular. Very many surfaces can be approximated has having a surface BRDF which is a combination of a Lambertian component and a specular component, which usually has some form of narrow lobe. Usually, the specular component is weighted by a **specular albedo**. Again, because specularities tend not to be examined in detail, the shape of this lobe is left unspecified. In this case, the surface *radiance* (because it must now depend on direction) in a given direction is typically approximated as:

$$L(\boldsymbol{x}, \theta_o, \phi_o) = \rho_d(\boldsymbol{x}) \int_{\Omega} L(\boldsymbol{x}, \theta_i, \phi_i) \cos\theta_i d\omega + \rho_s(\boldsymbol{x}) L(\boldsymbol{x}, \theta_s, \phi_s) \cos^n(\theta_s - \theta_o)$$

where θ_s, ϕ_s give the specular direction and ρ_s is the specular albedo. As we shall see, it is common not to reason about the exact magnitude of the specular radiance term.

Using this model implicitly excludes "too narrow" specular lobes, because most algorithms expect to encounter occasional small, compact specularities from light sources. Surfaces with too narrow specular lobes (mirrors) produce overwhelming quantities of detail in specularities. Similarly, "too broad" lobes are excluded because the specularities would be hard to identify.

1.4 Quick Reference: Radiometric Terminology for Light

Term	**Definition**	**Units**	**Application**
Radiance	the quantity of energy travelling at some point in a specified direction, per unit time, per unit area *perpendicular to the direction of travel*, per unit solid angle.	wm^2sr^{-1}	representing light travelling in free space; representing light reflected from a surface when the amount reflected depends strongly on direction
Irradiance	total incident power per unit surface area	wm^{-2}	representing light arriving at a surface
Radiosity	the total power leaving a point on a surface per unit area on the surface	wm^{-2}	representing light leaving a diffuse surface

1.5 Quick Reference: Radiometric Properties of Surfaces

Term	Definition	Units	Application
BRDF (Bidirectional Reflectance Distribution Function)	the ratio of the radiance in the outgoing direction to the incident irradiance	sr^{-1}	representing reflection off general surfaces where reflection depends strongly on direction
Directional Hemispheric Reflectance	the fraction of the incident irradiance in a given direction that is reflected by the surface, whatever the direction of reflection	unitless	representing reflection off a surface where direction is unimportant
Albedo	Directional hemispheric reflectance of a diffuse surface	unitless	representing a diffuse surface

1.6 Quick Reference: Important Types of Surface

Term	**Definition**	**Examples**
Diffuse surface; Lambertian surface	A surface whose BRDF is constant	Cotton cloth; many rough surfaces; many paints and papers; surfaces whose apparent brightness doesn't change with viewing direction
Specular surface	A surface that behaves like a mirror	Mirrors; polished metal
Specularity	Small bright patches on a surface that result from specular components of the BRDF	

1.7 Notes

We strongly recommend François Sillion's excellent book [**?**], for its very clear account of radiometric calculations. There are a variety of more detailed publications for reference [] Our discussion of reflection is thoroughly superficial. The specular plus diffuse model appears to be originally due to Cook, Torrance and Sparrow. A variety of modifications of this model appear in computer vision and computer graphics; see, for example []. Reflection models can be derived by combining a statistical description of surface roughness with electromagnetic considerations (e.g. []) or by adopting scattering models (e.g. [], where a surface is modelled by colourant particles embedded in a matrix, and a scattering model yields an approximate BRDF).

Top of the list of effects we omitted to discuss is off-specular glints, followed by specular backscatter. Off-specular glints commonly arise in brushed surfaces, where there is a large surface area oriented at a substantial angle to the macroscopic surface normal. This leads to a second specular lobe, due to this region. These effects can confuse algorithms that reason about shape from specularities, if the reasoning is close enough. Specular backscatter occurs when a surface reflects light back in the source direction — usually for a similar reason that off-specular glints occur. Again, the effect is likely to confuse algorithms that reason about shape from specularities.

It is commonly believed that rough surfaces are Lambertian. This belief has a substantial component of wishful thinking, because rough surfaces often have local shadowing effects that make the radiance reflected quite strongly dependent on the illumination angle. For example, a stucco wall illuminated at a near grazing angle shows a clear pattern of light and dark regions where facets of the surface face toward the light or are shadowed. If the same wall is illuminated along the normal, this pattern largely disappears. Similar effects at a finer scale are averaged to endow rough surfaces with measurable departures from a Lambertian model (for details, see [**?**; **?**; **?**; **?**]). Determining non-Lambertian models for surfaces that appear to be diffuse is a well established line of enquiry.

Another example of an object that does not support a simple macroscopic surface model is a field of flowers. A distant viewer should be able to abstract this field as a "surface"; however, doing so leads to a surface with quite strange properties. If one views such a field along a normal direction, one sees mainly flowers; a tangential view reveals both stalks and flowers, meaning that the colour changes dramatically (the effect is explored in []).

1.8 Assignments

Exercises

1. How many steradians in a hemisphere?

2. We have proven that radiance does not go down along a straight line *in a non-absorbing medium*, which makes it a useful unit. Show that if we were to

use power per square meter of foreshortened area (which is irradiance), the unit must change with distance along a straight line. How significant is this difference?

3. **An absorbing medium:** assume that the world is filled with an isotropic absorbing medium. A good, simple model of such a medium is obtained by considering a line along which radiance travels. If the radiance along the line is N at x, it will be $N - (\alpha dx)N$ at $x + dx$.
    - Write an expression for the radiance transferred from one surface patch to another in the presence of this medium.
    - Now *qualitatively* describe the distribution of light in a room filled with this medium, for α small and large positive numbers. The room is a cube, and the light is a single small patch in the center of the ceiling. Keep in mind that if α is large and positive, very little light will actually reach the walls of the room.
4. Identify common surfaces that are neither Lambertian nor specular, using the underside of a CD as a working example. There are a variety of important biological examples, which are often blue in colour. Give at least two different reasons that it could be advantageous to an organism to have a non-Lambertian surface.
5. Show that for an ideal diffuse surface the directional hemispheric reflectance is constant; now show that if a surface has constant directional hemispheric reflectance, it is ideal diffuse.
6. Show that the BRDF of an ideal specular surface is

$$\rho_{bd}(\theta_o, \phi_o, \theta_i, \phi_i) = \rho_s(\theta_i)\{2\delta(\sin^2\theta_o - \sin^2\theta_i)\}\{\delta(\phi_o - \phi\pi)\}$$

    where $\rho_s(\theta_i)$ is the fraction of radiation that leaves.
7. Why are specularities brighter than diffuse reflection?
8. A surface has constant BRDF. What is the maximum possible value of this constant? Now assume that the surface is known to absorb 20% of the radiation incident on it (the rest is reflected); what is the value of the BRDF?
9. The eye responds to radiance. Explain why Lambertian surfaces are often referred to as having a brightness that is independent of viewing angle.
10. Show that the solid angle subtended by a sphere of radius ϵ at a point a distance r away from the center of the sphere is approximately $\pi(\frac{\epsilon}{r})^2$, for $r \gg \epsilon$.

Chapter 2

SOURCES, SHADOWS AND SHADING

We shall start by describing the basic radiometric properties of various light sources. We shall then develop models of source geometries and discuss the radiosity and the shadows that result from these sources. The purpose of all this physics is to establish a usable model of the shading on a surface; we develop two kinds of model in some detail. We show that, when one of these models applies, it is possible to extract a representation of the shape and albedo of an object from a series of images under different lights. Finally, we describe the effects that result when surfaces reflect light onto one another.

2.1 Radiometric Properties of Light Sources

Anything that emits light is a light source. To describe a source, we need a description of the radiance it emits in each direction. Typically, emitted radiance is dealt with separately from reflected radiance. Together with this, we need a description of the geometry of the source, which has profound effects on the spatial variation of light around the source and on the shadows cast by objects near the source. Sources are usually modelled with quite simple geometries, for two reasons: firstly, many synthetic sources can be modelled as point sources or line sources fairly effectively; secondly, sources with simple geometries can still yield surprisingly complex effects.

We seldom need a complete description of the spectral radiance a source emits in each direction. It is more usual to model sources as emitting a constant radiance in each direction, possibly with a family of directions zeroed (like a spotlight). The proper quantity in this case is the **exitance**, defined as

> the internally generated energy radiated per unit time and per unit area on the radiating surface (after [**?**])

Exitance is similar to radiosity, and can be computed as

$$E(\boldsymbol{x}) = \int_{\Omega} L_e(\boldsymbol{x}, \theta_o, \phi_o) \cos\theta_o d\omega$$

In the case of a coloured source, one would use spectral exitance or spectral radiance as appropriate. Sources can have radiosity as well as exitance, because energy may be reflected off the source as well as generated within it.

2.2 Qualitative Radiometry

We should like to know how "bright" surfaces are going to be under various lighting conditions, and how this "brightness" depends on local surface properties, on surface shape, and on illumination. The most powerful tool for analysing this problem is to think about *what a source looks like from the surface.* In some cases, this technique us to give qualitative descriptions of "brightness" without knowing what the term means.

Recall from section 1.1.1 and figure 1.1 that a surface patch sees the world through a hemisphere of directions at that patch. The radiation arriving at the surface along a particular direction passes through a point on the hemisphere. If two surface patches have equivalent incoming hemispheres, they must have the same incoming radiation, *whatever the outside world looks like.* This means that any difference in "brightness" between patches with the same incoming hemisphere is a result of different surface properties.

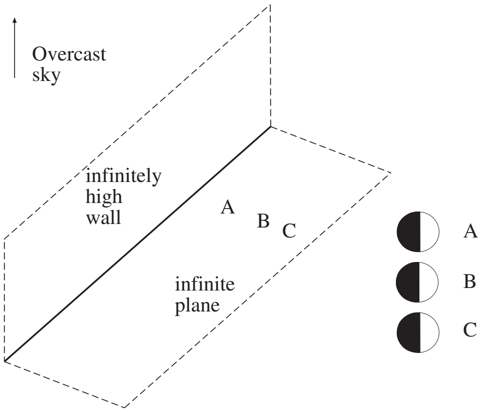


Figure 2.1. A geometry in which a qualitative radiometric solutions can be obtained by thinking about what the world looks like from the point of view of a patch. We wish to know what the brightness looks like at the base of two different infinitely high walls. In this geometry, an infinitely high matte black wall cuts off the view of the overcast sky — which is a hemisphere of infinite radius and uniform "brightness". On the right, we show a representation of the directions that see or do not see the source at the corresponding points, obtained by flattening the hemisphere to a circle of directions (or, equivalently, by viewing it from above). Since each point has the same input hemisphere, the brightness must be uniform.

Lambert determined the distribution of "brightness" on a uniform plane at the base of an infinitely high black wall illuminated by an overcast sky (see Figure 2.1). In this case, every point on the plane must see the same hemisphere — half of its viewing sphere is cut off by the wall, and the other half contains the sky, which is uniform — and the plane is uniform, so every point must have the same "brightness".

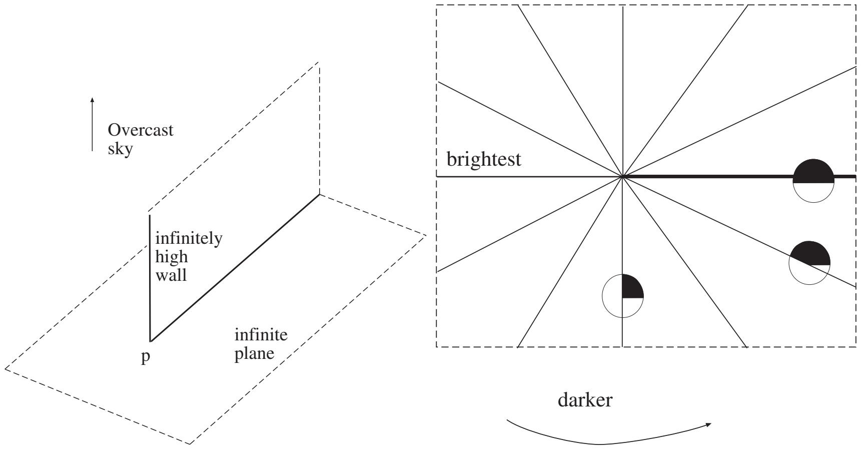


Figure 2.2. We now have a matte black, infinitely thin, half-infinite wall on an infinite white plane. This geometry also sees an overcast sky of infinite radius and uniform "brightness". In the text, we show how to determine the curves of similar "brightness" on the plane. These curves are shown on the right, depicted on an overhead view of the plane; the thick line represents the wall. Superimposed on these curves is a representation of the input hemisphere for some of these isophotes. Along these curves, the hemisphere is fixed (by a geometrical argument), but they change as one moves from curve to curve.

A second example is somewhat trickier. We now have an infinitely thin black wall that is infinitely long only in one direction, on an infinite plane (Figure 2.2). A qualitative description would be to find the curves of equal "brightness" look like. It is fairly easy to see that all points on any line passing through the point $\boldsymbol{p}$ in Figure 2.2 see the same input hemisphere, and so must have the same "brightness". Furthermore, the distribution of "brightness" on the plane must have a symmetry about the line of the wall — we expect the brightest points to be along the extension of the line of the wall, and the darkest to be at the base of the wall.

2.3 Sources and their Effects

There are three main types of geometrical source models: point sources, line sources and area sources. In this section, we obtain expressions for radiosity sources of these types produce at a surface patch. These expressions could be obtained by thinking about the limiting behaviour of various nasty integrals. Instead, we obtain them by thinking about the appearance of the source from the patch.

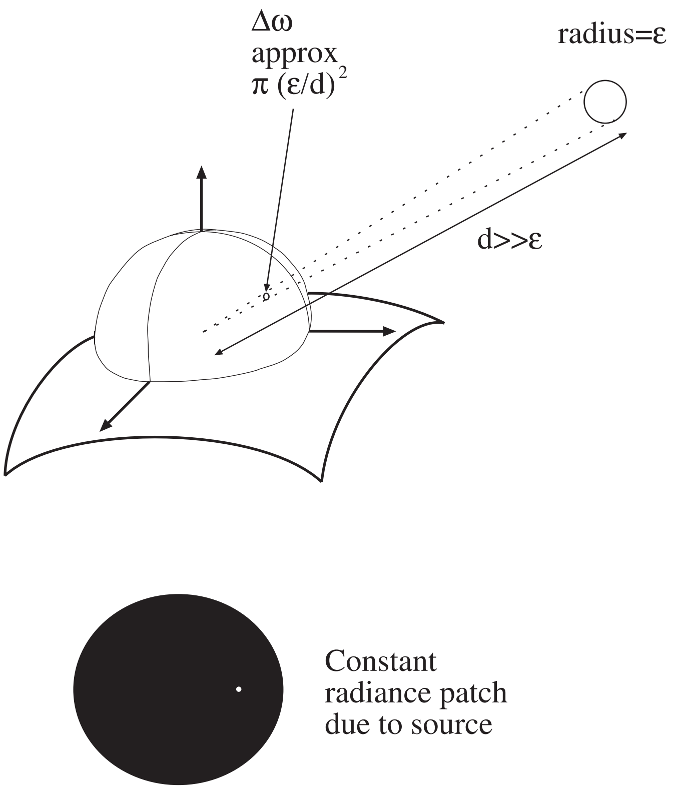


Figure 2.3. A surface patch sees a distant sphere of small radius; the sphere produces a small illuminated patch on the input hemisphere of the sphere. In the text, by reasoning about the scaling behaviour of this patch as the distant sphere moves further away or gets bigger, we obtain an expression for the behaviour of the point source.

2.3.1 Point Sources

A common approximation is to consider a light source as a point. It is a natural model to use, because many sources are physically small compared to the environment in which they stand. We can obtain a model for the effects of a point sources by modelling the source as a very small sphere which emits light at each point on the sphere, with an exitance that is constant over the sphere.

Assume that a surface patch is viewing a sphere of radius ϵ, at a distance r away, and that $\epsilon \gg r$. We assume the sphere is far away from the patch relative to its radius (a situation that almost always applies for real sources). Now the solid angle that the source subtends is Ω_s. This will behave approximately proportional to

$$\frac{\epsilon^2}{r^2}$$

The pattern of illumination that the source creates on the hemisphere will (roughly) scale, too. As the sphere moves away, the rays leaving the surface patch and striking the sphere move closer together (roughly) evenly, and the collection changes only slightly (a small set of new rays is added at the rim — the contribution from these rays must be very small, because they come from directions tangent to the sphere). In the limit as ϵ tends to zero, no new rays are added.

The radiosity due to the source is obtained by integrating this pattern, times $\cos\theta_i$ over the patch of solid angle. As ϵ tends to zero, the patch shrinks and the $\cos\theta_i$ is close to constant. If ρ is the surface albedo, all this means the expression for radiosity due to the point source will be

$$\rho\left(\frac{\epsilon}{r}\right)^2 E\cos\theta$$

where E is a term in the exitance of the source, integrated over the small patch. We don't need a more detailed expression for E (to determine one, we would need to actually do the integral we have shirked).

A Nearby Point Source

The angle term can be written in terms of $\boldsymbol{N}(\boldsymbol{x})$ (the unit normal to the surface) and $\boldsymbol{S}(\boldsymbol{x})$ (a vector from $\boldsymbol{x}$ to the source, whose length is E) to yield the standard **nearby point source model:**

$$\rho_d(\boldsymbol{x})\frac{\boldsymbol{N}(\boldsymbol{x})\cdot\boldsymbol{S}(\boldsymbol{x})}{r(\boldsymbol{x})^2}$$

This is an extremely convenient model, because it gives an explicit relationship between radiosity and shape (the normal term). In this model, $\boldsymbol{S}$ is usually called the **source vector**. It is common (and incorrect!) to omit the dependency on distance to the source from this model.

A Point Source at Infinity

The sun is far away; as a result, the terms $1/r(\boldsymbol{x})^2$ and $\boldsymbol{S}(\boldsymbol{x})$ are essentially constant. In this case, the point source is referred to as being a **point source at infinity**. If all the surface patches we are interested in are close together with respect to the distance to the source, $r(\boldsymbol{x}) = r_0 + \Delta r(\boldsymbol{x})$ where $r_0 \gg \Delta r(\boldsymbol{x})$. Furthermore, $\boldsymbol{S}(\boldsymbol{x}) = \boldsymbol{S}_0 + \Delta\boldsymbol{S}(\boldsymbol{x})$, where $\mid \boldsymbol{S}_0 \mid \gg \mid \Delta\boldsymbol{S}(\boldsymbol{x}) \mid$. We now have that:

$$\frac{\boldsymbol{N}\cdot\boldsymbol{S}(\boldsymbol{x})}{r(\boldsymbol{x})^2} = \frac{\boldsymbol{N}\cdot(\boldsymbol{S}_0+\Delta\boldsymbol{S}(\boldsymbol{x}))}{(r_0+\Delta r(\boldsymbol{x}))^2} = \frac{\boldsymbol{N}\cdot\boldsymbol{S}_0}{r_0^2}\left[1-2\frac{(\boldsymbol{N}\cdot\Delta\boldsymbol{S}(\boldsymbol{x}))\Delta r\boldsymbol{x}}{r_0}+\ldots\right] \approx \frac{\boldsymbol{N}\cdot\boldsymbol{S}_0}{r_0^2}$$

so that our model for the radiosity due to a point source at infinity becomes:

$$B(\boldsymbol{x}) = \rho_d(\boldsymbol{x})(\boldsymbol{N}\cdot\boldsymbol{S})$$

Choosing a Point Source Model

A point source at infinity is a good model for the sun, for example, because the solid angle that the sun subtends is small and essentially constant, wherever it appears in the field of view (this test means that our approximation step is valid). Usually, we do not care about the structure of the term $\boldsymbol{S}$, which is again known as the **source vector**. If we use linear sensors with an unknown gain, for example, we can roll the source intensity and the unknown gain into this term.

As you should expect from the derivation, this is a poor model *when the distance between objects is similar in magnitude to the distance to the source*. In this case, we cannot use the series approximation to pretend that the radiosity due to the source does not go down with distance to the source.

The heart of the problem is easy to see if we consider what the source looks like from different surface patches. It must look bigger to nearer surface patches (however small its radius); this means that the radiosity due to the source must go up. If the source is sufficiently distant — for example, the sun — we can ignore this effect because the source does not change in apparent size for any plausible motion.

However, for configurations like a light bulb in the center of a room the solid angle subtended by the source goes up as the inverse square of the distance, meaning that the radiosity due to the source will do so, too. The correct model to use in this case is the point source of Section 2.3.1. The difficulty with this model is that radiosity changes very sharply over space, in a way that is inconsistent with experience. For example, if a point source is placed at the center of a cube, then the radiosity in the corners is roughly a ninth that at the center of each face — but the corners of real rooms are nowhere near as dark as that. The explanation must wait until we have discussed shading models.

2.3.2 Line Sources

A line source has the geometry of a line — a good example is a single fluorescent light bulb. Line sources are not terribly common in natural scenes or in synthetic environments, and we will discuss them only briefly. Their main interest is as an example for radiometric problems; in particular, the radiosity of patches reasonably close to a line source changes as the reciprocal of distance to the source (rather than the square of the distance). The reasoning is more interesting than the effect. We model a line source as a thin cylinder with diameter ϵ. Assume for the moment that the line source is infinitely long, and that we are considering a patch that views the source frontally, as in Figure 2.4.

Figure 2.4 sketches the appearance of the source from the point of view of patch 1; now move the patch closer, and consider patch 2 — the width of the region on the hemisphere corresponding to the source changes, but not the length (because the source is infinitely long). In turn, because the width is approximately ϵ/r, the radiosity due to the source must go down with the reciprocal of distance. It is easy to see that with a source that is not infinitely long, this applies as long as the patch is reasonably close.

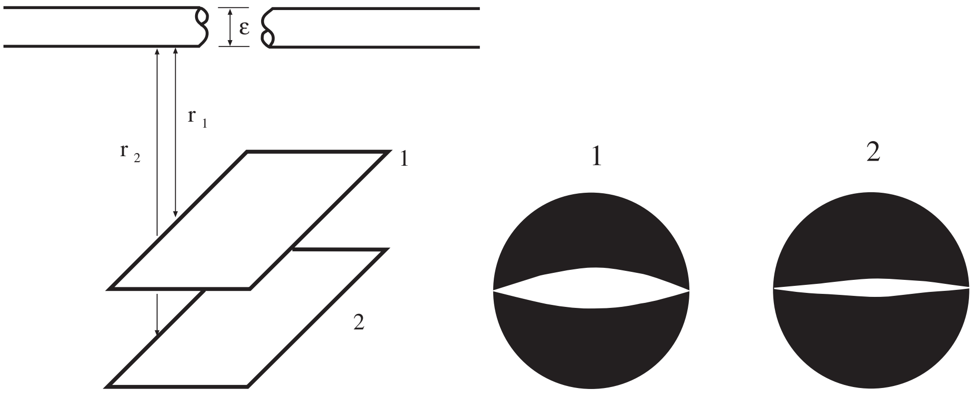


Figure 2.4. The radiosity due to a line source goes down as the reciprocal of distance, for points that are reasonably close to the source. On the left, two patches viewing an infinitely long, narrow cylinder with constant exitance along its surface and diameter ϵ. On the right, the view of the source *from each patch*, drawn as the underside of the input hemisphere seen from below. Notice that the length of the source on this hemisphere does not change, but the width does (as ϵ/r). This yields the result.

2.3.3 Area Sources

Area sources are important, for two reasons. Firstly, they occur quite commonly in natural scenes — an overcast sky is a good example — and in synthetic environments — for example, the fluorescent light boxes found in many industrial ceilings. Secondly, a study of area sources will allow us to explain various shadowing and interreflection effects. Area sources are normally modelled as surface patches whose emitted radiance is independent of position and of direction — they can be described by their exitance.

An argument similar to that used for line sources shows that, for points not too distant from the source, the radiosity due to an area source does not change with distance to the source. This explains the widespread use of area sources in illumination engineering — they generally yield fairly uniform illumination. For our applications, we need a more exact description of the radiosity due to an area source, so we will need to write out the integral.

The Exact Radiosity due to an Area Source

Assume we have a diffuse surface patch which is illuminated by an area source with exitance $E(\boldsymbol{u})$. We use $\boldsymbol{u}$ as a coordinate on the source, and instead of writing angles in coordinates, we write $\boldsymbol{u} \to \boldsymbol{x}$ for the direction from $\boldsymbol{u}$ to $\boldsymbol{x}$ (more notation is illustrated in Figure 2.5). The radiosity on the surface is obtained by summing the incoming radiance over all incoming directions. This integral can be transformed into an integral over the source (by turning the angle to a source point into a coordinate on the source). The process looks like:

$$
\begin{aligned}
B(\boldsymbol{x}) &= \rho_d(\boldsymbol{x}) \int_{\Omega} L_i(\boldsymbol{x}, \boldsymbol{u} \to \boldsymbol{x}) \cos\theta_i d\omega \\
&= \rho_d(\boldsymbol{x}) \int_{\Omega} L_e(\boldsymbol{u}, \boldsymbol{u} \to \boldsymbol{x}) \cos\theta_i d\omega \\
&= \rho_d(\boldsymbol{x}) \int_{\Omega} (\frac{1}{\pi} E(\boldsymbol{u})) \cos\theta_i d\omega \\
&= \rho_d(\boldsymbol{x}) \int_{Source} (\frac{1}{\pi} E(\boldsymbol{u})) \cos\theta_i (\cos\theta_s \frac{dA_{\boldsymbol{u}}}{r^2}) \\
&= \rho_d(\boldsymbol{x}) \int_{Source} E(\boldsymbol{u}) \frac{\cos\theta_i \cos\theta_s}{\pi r^2} dA_{\boldsymbol{u}}
\end{aligned}
$$

The transformation works because radiance is constant along straight lines and because $E(\boldsymbol{u}) = 1/\pi L_e(\boldsymbol{u})$. This transformation is very useful, because it means we do not have to worry about consistent angular coordinate systems, but these integrals are still almost always impossible to do in closed form (but see exercise).

2.4 Local Shading Models

We have studied the physics of light because we want to know how bright things will be, and why, in the hope of extracting object information from these models. Currently, we know the radiosity at a patch *due to a source* but this is not a shading model. Radiance could arrive at surface patches in other ways (it could, for example, be reflected from other surface patches); we need to know which components to account for.

This topic is fraught with all the difficulties involved in choosing a model. The easiest model to manipulate is a **local shading model**, which models the radiosity at a surface patch as the sum of the radiosity due to sources and sources alone. This model will support a variety of algorithms and theories (see section 2.5). Unfortunately, this model often produces wildly inaccurate predictions. Even worse, there are is little reliable information about when this model is safe to use.

An alternate model is to account for all radiation (section 2.6). This takes into account radiance arriving from sources, and that arriving from radiating surfaces. This model is physically accurate, but usually very hard to manipulate.

2.4.1 Local Shading Models for Point Sources

The local shading model for a set of point sources is:

$$
B(\boldsymbol{x}) = \sum_{s \in \text{sources visible from } \boldsymbol{x}} B_s(\boldsymbol{x})
$$

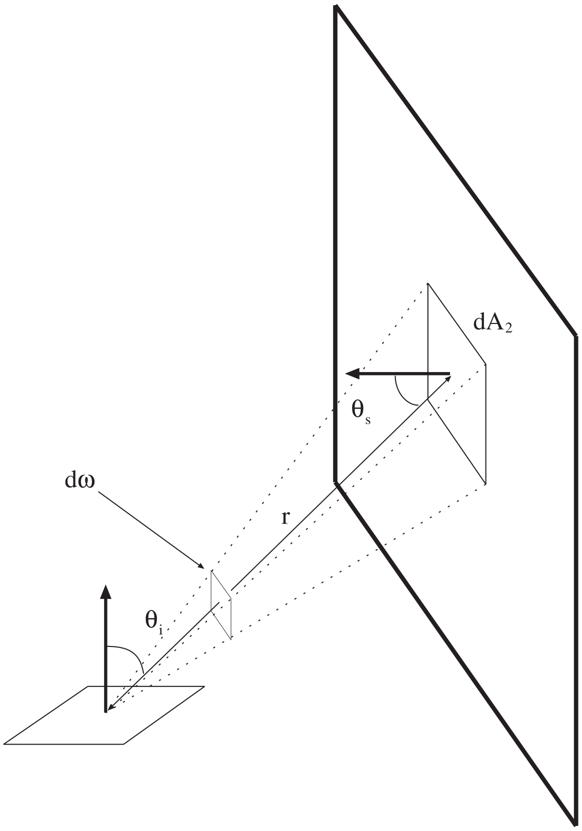


Figure 2.5. A diffuse source illuminates a diffuse surface. The source has exitance $E(\boldsymbol{u})$ and we wish to compute the radiosity on the patch due to the source. We do this by transforming the integral of incoming radiance at the surface into an integral over the source area. This transformation is convenient, because it avoids us having to use different angular domains for different surfaces; however, it still leads to an integral that is usually impossible in closed form.

where $B_s(\boldsymbol{x})$ is the radiosity due to source s. This expression is fairly innocuous; but notice that if all the sources are point sources at infinity, the expression becomes:

$$B(\boldsymbol{x}) = \sum_{s \in \text{sources visible from } \boldsymbol{x}} \rho_d(\boldsymbol{x}) \boldsymbol{N}(\boldsymbol{x}) \cdot \boldsymbol{S}_s$$

so that if we confine our attention to a region where all points can see the same sources, we could add all the source vectors to obtain a single virtual source that had the same effects. The relationship between shape and shading is pretty direct here — the radiosity is a measurement of one component of the surface normal.

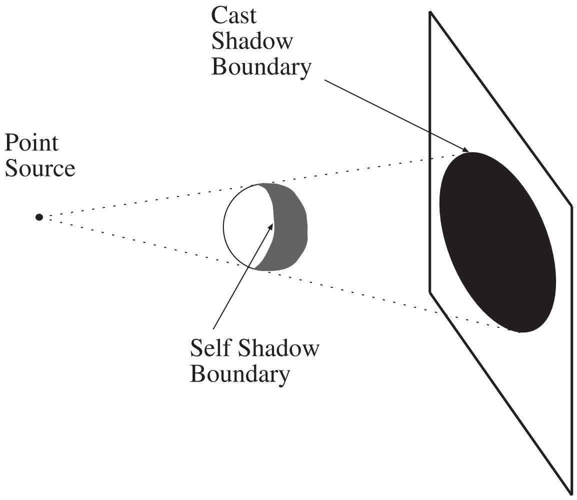


Figure 2.6. Shadows cast by point sources on planes are relatively simple. Self shadow boundaries occur when the surface turns away from the light and cast shadow boundaries occur when a distant surface occludes the view of the source.

For point sources that are not at infinity the model becomes:

$$B(\boldsymbol{x}) = \sum_{s \in \text{sources visible from } \boldsymbol{x}} \rho_d(\boldsymbol{x}) \frac{\boldsymbol{N}(\boldsymbol{x}) \cdot \boldsymbol{S}(\boldsymbol{x})}{r_s(\boldsymbol{x})^2}$$

where $r_s(\boldsymbol{x})$ is the distance from the source to $\boldsymbol{x}$; the presence of this term means that the relationship between shape and shading is somewhat more obscure.

The Appearance of Shadows

In a local shading model, shadows occur when the patch can not see one or more sources. In this model, point sources produce a series of shadows with crisp boundaries; shadow regions where no source can be seen are particularly dark. Shadows cast with a single source can be very crisp and very black, depending on the size of the source and the albedo of other nearby surfaces (which reflect light, whether we model the effect or not!). It was a popular 19'th century pastime to cast such shadows onto paper, and then draw them, yielding the silhouettes which are still occasionally to be found in antiques shops.

The geometry of the shadow cast by a point source on a plane is analogous to the geometry of viewing in a perspective camera (Figure 2.6). Any patch on the plane is in shadow if a ray from the patch to the source passes through an object. This means that there are two kinds of shadow boundary. At **self shadow boundaries**, the surface is turning away from the light, and a ray from the patch to the source is tangent to the surface. At **cast shadow boundaries**, from the perspective of the patch, the source suddenly disappears behind an occluding object. Shadows cast onto curved surfaces can have extremely complex geometries, however.

If there are many sources, the shadows will be less dark (except at points where no source is visible) and there can be very many qualitatively distinct shadow regions (each source casts its own shadow — some points may not see more than one source). One example of this effect occurs in televised soccer matches — because the stadium has multiple bright distant point-like illuminants spaced evenly around the perimeter of the stadium, there is a set of shadows radiating evenly around each player's feet. These shadows typically become brighter or darker as the player moves around, usually because the illumination due to other sources and to interreflections in the region of the shadow increases or decreases.

2.4.2 Area Sources and their Shadows

The local shading model for a set of area sources is significantly more complex, because it is possible for patches to see only a portion of a given source. The model becomes:

$$\begin{aligned} B(\boldsymbol{x}) &= \sum_{s \in \text{all sources}} \left\{ \int_{\text{visible component of source } s} \text{Radiosity due to source} \right\} \\ &= \sum_{s \in \text{all sources}} \int_{\text{visible component of source } s} \left\{ E(\boldsymbol{u}) \frac{\cos\theta_u \cos\theta_s}{\pi r^2} dA_{\boldsymbol{u}} \right\} \end{aligned}$$

using the terminology of Figure 2.5; usually, we assume that E is constant over the source.

Area sources do not produce dark shadows with crisp boundaries. This is because, from the perspective of a viewing patch, the source appears slowly from behind the occluding object (think of an eclipse of the moon — it is an exact analogy). It is common to distinguish between points in the **umbra** (a Latin word, meaning "shadow") — which cannot see the source at all — and points in the **penumbra** (a compound of Latin words, meaning "almost shadow") — which see part of the source. The vast majority of indoor sources are area sources of one form or another, so the effects are quite easy to see; hold an arm quite close to the wall (for reasons we will discuss below) and look at the shadow it casts — there is a dark core, which gets larger as the arm gets closer to the wall; this is the umbra — surrounded by a lighter region with a fuzzier boundary (the penumbra). Figure 2.7 illustrates the geometry.

2.4.3 Ambient Illumination

One problem with local shading models should be apparent immediately; they predict that some shadow regions are arbitrarily dark, because they cannot see the source. This prediction is inaccurate in almost every case, because shadows are illuminated by light from other diffuse surfaces. This effect can be very significant.

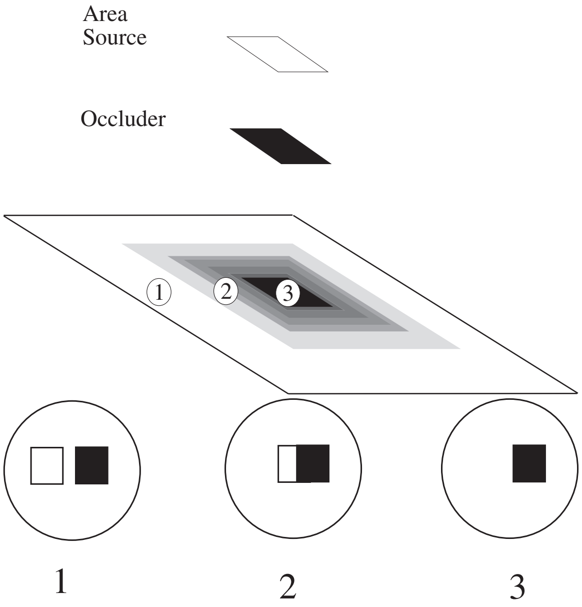


Figure 2.7. Area sources generate complex shadows with smooth boundaries, because, from the point of view of a surface patch, the source disappears slowly behind the occluder. Regions where the source cannot be seen at all are known as the *umbra*; regions where some portion of the source is visible are known as the *penumbra*. A good model is to imagine lying with your back to the surface, looking at the world above. At point 1, you can see all of the source; at point 2, you can see some of it; and at point 3 you can see none of it.

In rooms with light walls and area sources, it is possible to see shadows only by holding objects close to the wall or close to the source. This is because a patch on the wall sees all the other walls in the room; and until an object is close to the wall, it blocks out only a very small fraction of each patches visual hemisphere.

For some environments, the total irradiance a patch obtains from other patches is roughly constant and roughly uniformly distributed across the input hemisphere. This must be true for the interior of a sphere with a constant distribution of radiosity (by symmetry), and (by accepting a model of a cube as a sphere) is roughly true for the interior of a room with white walls. In such an environment it is sometimes possible to model the effect of other patches by adding an **ambient illumination term** to each patch's radiosity. There are two strategies for determining this term. Firstly, if each patch sees the same proportion of the world (for example, the interior of a sphere), we can add the same constant term to the radiosity of each patch. The magnitude of this term is usually guessed.

Secondly, if some patches see more or less of the world than others (this happens if regions of the world occlude a patch's view, for example, a patch at the bottom

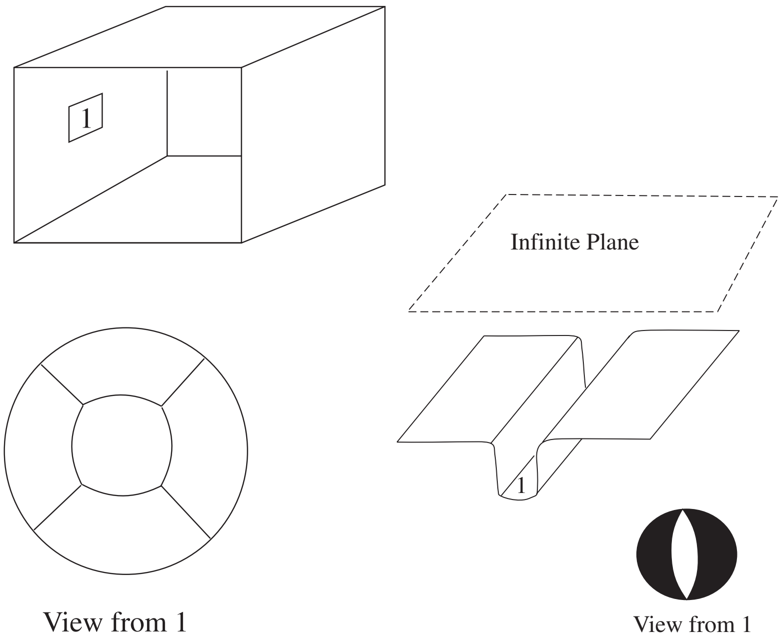


Figure 2.8. Ambient illumination is a term added to the radiosity predictions of local shading models to model the effects of radiosity from distant, reflecting surfaces. In a world like the interior of a sphere or of a cube (the case on the left), where a patch sees roughly the same thing from each point, a constant ambient illumination term is often acceptable. In more complex worlds, some surface patches see much less of the surrounding world than others. For example, the patch at the base of the groove on the right sees relatively little of the outside world, which we model as an infinite polygon of constant exitance; its input hemisphere is shown below.

of a groove), this can be taken into account. To do so, we need a model of the world *from the perspective of the patch under consideration.* A natural strategy is to model the world as a large, distant polygon of constant radiosity, where the view of this polygon is occluded at some patches (see Figure 2.8). The result is that the ambient term is smaller for patches that see less of the world. This model is often more accurate than adding a constant ambient term. Unfortunately, it is much more difficult to extract information from this model, possibly as difficult as for a global shading model.

2.5 Application: Photometric Stereo

We will reconstruct a patch of surface from a series of pictures of the surface, taken under different illuminants. First, we need a camera model. For simplicity, we choose a camera situated so that the point (x, y, z) in space is imaged to the point

(x, y) in the camera (the method we describe will work for the other camera models described in chapter **??**).

In this case, to measure the shape of the surface we need to obtain the depth to the surface. This suggests representing the surface as $(x, y, f(x, y))$ — a representation known as a **Monge patch**, after a French military engineer who first used it (figure 2.9). This representation is attractive, because we can determine a unique point on the surface by giving the image coordinates. Notice that to obtain a measurement of a solid object, we would need to reconstruct more than one patch, because we need to observe the back of the object.

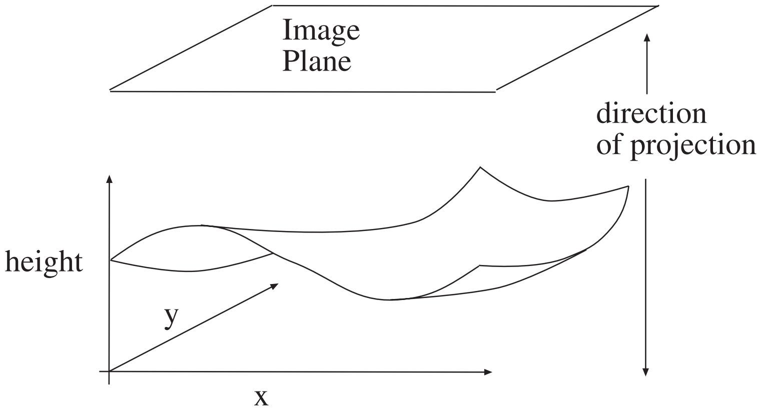


Figure 2.9. A Monge patch is a representation of a piece of surface as a height function. For the photometric stereo example, we assume that an orthographic camera — one that maps (x, y, z) in space to (x, y) in the camera — is viewing a Monge patch. This means that the shape of the surface can be represented as a function of position in the image.

Photometric stereo is a method for recovering a representation of the Monge patch from image data. The method involves reasoning about the image intensity values for several different images of a surface in a fixed view, illuminated by different sources. This method will recover the height of the surface at points corresponding to each pixel; in computer vision circles, the resulting representation is often known as a **height map**, **depth map** or **dense depth map**.

Fix the camera and the surface in position, and illuminate the surface using a point source that is far away compared to the size of the surface. We adopt a local shading model and assume that there is no ambient illumination — more about this later — so that the radiosity at a point $\boldsymbol{x}$ on the surface is

$$B(\boldsymbol{x}) = \rho(\boldsymbol{x})\boldsymbol{N}(\boldsymbol{x}) \cdot \boldsymbol{S}_1$$

where $\boldsymbol{N}$ is the unit surface normal and $\boldsymbol{S}_1$ is the source vector. We can write $B(x, y)$ for the radiosity of a point on the surface, because there is only one point on the surface corresponding to the point (x, y) in the camera. Now we assume that the response of the camera is linear in the surface radiosity, and so have that the value of a pixel at (x, y) is

$$\begin{aligned} I(x, y) &= kB(\boldsymbol{x}) \\ &= kB(x, y) \\ &= k\rho(x, y)\boldsymbol{N}(x, y) \cdot \boldsymbol{S}_1 \\ &= \boldsymbol{g}(x, y) \cdot \boldsymbol{V}_1 \end{aligned}$$

where $\boldsymbol{g}(x, y) = \rho(x, y)\boldsymbol{N}(x, y)$ and $\boldsymbol{V}_1 = k\boldsymbol{S}_1$, where k is the constant connecting the camera response to the input radiance.

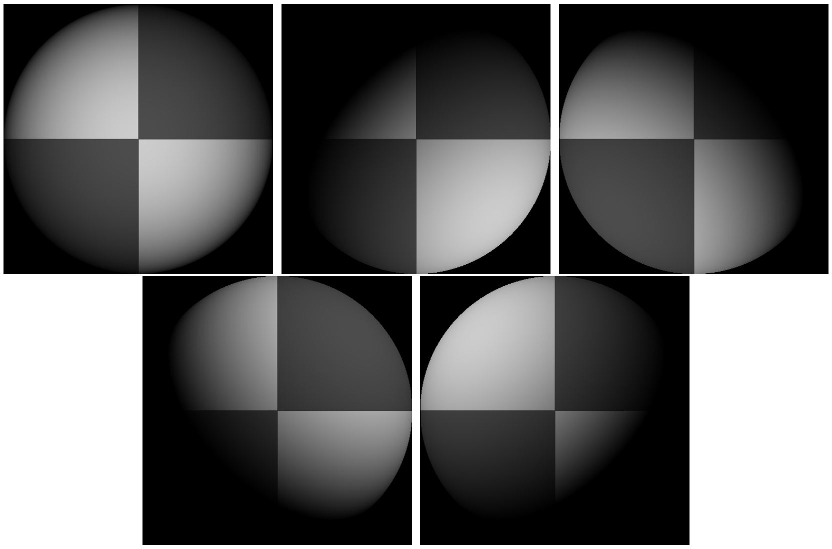


Figure 2.10. Five synthetic images of a sphere, all obtained in an orthographic view from the same viewing position. These images are shaded using a local shading model and a distant point source. This is a convex object, so the only view where there is no visible shadow occurs when the source direction is parallel to the viewing direction. The variations in brightness occuring under different sources code the shape of the surface.

In these equations, $\boldsymbol{g}(x, y)$ describes the surface and $\boldsymbol{V}_1$ is a property of the illumination and of the camera. We have a dot-product between a vector field $\boldsymbol{g}(x, y)$ and a vector $\boldsymbol{V}_1$ which could be measured; with enough of these dot-products, we could reconstruct $\boldsymbol{g}$, and so the surface.

2.5.1 Normal and Albedo from Many Views

Now if we have n sources, for each of which $\boldsymbol{V}_i$ is known, we stack each of these $\boldsymbol{V}_i$ into a known matrix $\mathcal{V}$, where

$$\mathcal{V} = \begin{pmatrix} \boldsymbol{V}_1^T \\ \boldsymbol{V}_2^T \\ \ldots \\ \boldsymbol{V}_n^T \end{pmatrix}$$

For each image point, we stack the measurements into a vector

$$\boldsymbol{i}(x, y) = \{I_1(x, y), I_2(x, y), \ldots, I_n(x, y)\}^T$$

Notice we have one vector per image point; each vector contains all the image brightnesses observed at that point for different sources. Now we have

$$\boldsymbol{i}(x, y) = \mathcal{V}\boldsymbol{g}(x, y)$$

and $\boldsymbol{g}$ is obtained by solving this linear system — or rather, one linear system per point in the image. Typically, $n > 3$ so that a least squares solution is appropriate. This has the advantage that the residual error in the solution provides a check on our measurements.

The difficulty with this approach is that substantial regions of the surface may be in shadow for one or the other light (see figure 2.10). There is a simple trick that deals with shadows. If there really is no ambient illumination, then we can form a matrix from the image vector and multiply both sides by this matrix; this will zero out any equations from points that are in shadow. We form

$$\mathcal{I}(x, y) = \begin{pmatrix} I_1(x, y) & \ldots & 0 & 0 \\ 0 & I_2(x, y) & \ldots & 0 \\ \ldots & & & \\ 0 & 0 & \ldots & I_n(x, y) \end{pmatrix}$$

and

$$\mathcal{I}\boldsymbol{i} = \mathcal{I}\mathcal{V}\boldsymbol{g}(x, y)$$

and $\mathcal{I}$ has the effect of zeroing the contributions from shadowed regions, because the relevant elements of the matrix are zero at points that are in shadow. Again, there is one linear system per point in the image; at each point, we solve this linear system to recover the $\boldsymbol{g}$ vector at that point. Figure 2.11 shows the vector field $\boldsymbol{g}(x, y)$ recovered from the images of figure 2.10.

Measuring Albedo

We can extract the albedo from a measurement of $\boldsymbol{g}$, because $\boldsymbol{N}$ is the unit normal. This means that $\mid \boldsymbol{g}(x, y) \mid = \rho(x, y)$. This provides a check on our measurements as well. Because the albedo is in the range zero to one, any pixels where $\mid \boldsymbol{g} \mid$ is greater than one are suspect — either the pixel is not working, or $\mathcal{V}$ is incorrect. Figure 2.12 shows albedo recovered using this method for the images of figure 2.10.

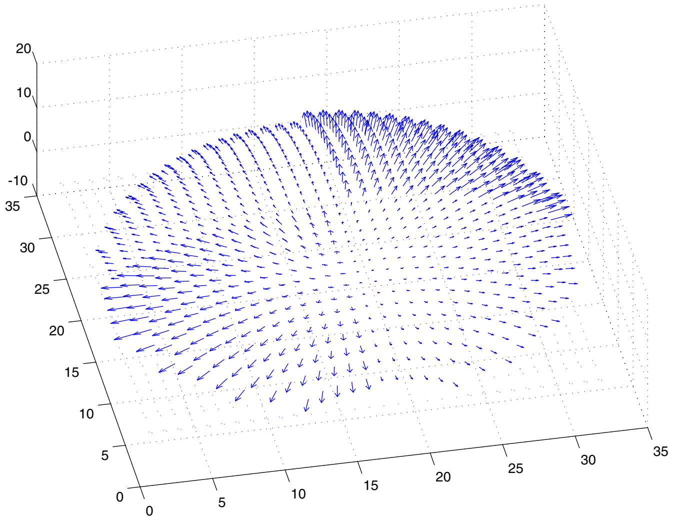


Figure 2.11. The vector field $\boldsymbol{g}(x, y)$ recovered from the input data of 2.10, mapped onto the recovered surface (this just makes the vectors more visible. The vector field is shown for every 16'th pixel in each direction, to reduce the complexity of the plot and make the structure clear.

Recovering Normals

We can extract the surface normal from $\boldsymbol{g}$, because the normal is a unit vector

$$\boldsymbol{N}(x, y) = \frac{\boldsymbol{g}(x, y)}{|\,\boldsymbol{g}(x, y)\,|}$$

Figure 2.13 shows normal values recovered for the images of figure 2.10.

2.5.2 Shape from Normals

The surface is $(x, y, f(x, y))$, so the normal as a function of (x, y) is

$$\boldsymbol{N}(x, y) = \frac{1}{\sqrt{1 + \frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2}} \left\{ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, 1 \right\}^T$$

To recover the depth map, we need to determine $f(x, y)$ from measured values of the unit normal.

Assume that the measured value of the unit normal at some point (x, y) is $(a(x, y), b(x, y), c(x, y))$. Then we have that

$$\frac{\partial f}{\partial x} = \frac{a(x, y)}{c(x, y)} \text{ and } \frac{\partial f}{\partial y} = \frac{b(x, y)}{c(x, y)}$$

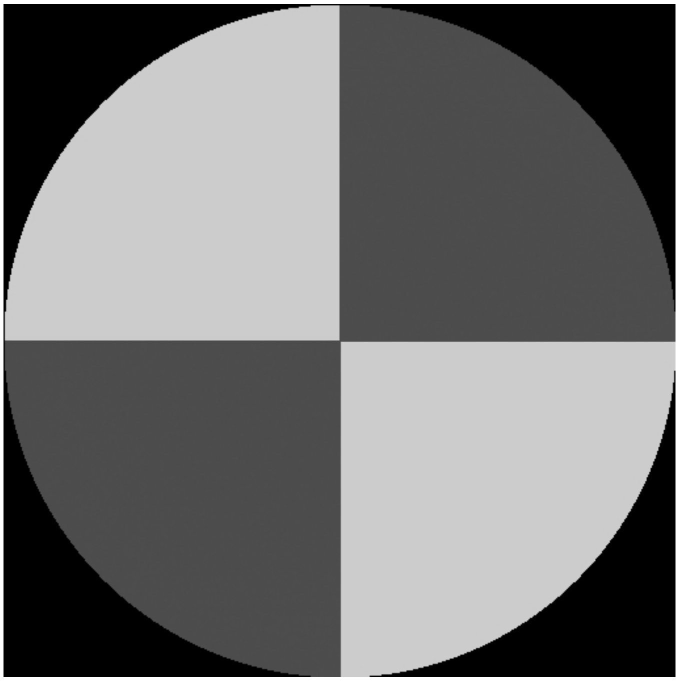


Figure 2.12. The magnitude of the vector field $\boldsymbol{g}(x,y)$ recovered from the input data of 2.10 represented as an image — this is the reflectance of the surface.

We have another check on our data set, because

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$$

so that we expect that

$$\frac{\partial \left(\frac{a(x,y)}{c(x,y)} \right)}{\partial y} - \frac{\partial \left(\frac{b(x,y)}{c(x,y)} \right)}{\partial x}$$

should be small at each point; in principle it should be zero, but we would have to estimate these partial derivatives numerically, and so should be willing to accept small values. This test is known as a test of **integrability**, which in vision applications always boils down to checking that first partials are equal.

Shape by Integration

Assuming that the partial derivatives pass this sanity test, we can reconstruct the surface up to some constant depth error. The partial derivative gives the change in surface height with a small step in either the x or the y direction. This means

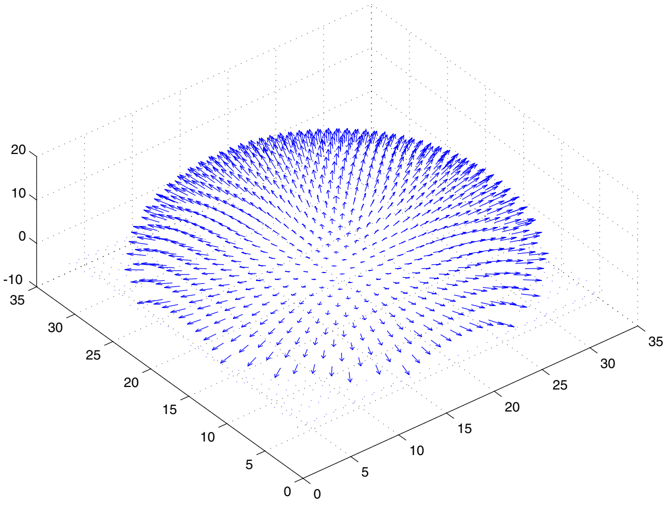


Figure 2.13. The normal field recovered from the surface of figure 2.10.

we can get the surface by summing these changes in height along some path. In particular, we have that

$$f(x,y) = \oint_C \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \cdot d\boldsymbol{l} + c$$

where C is a curve starting at some fixed point and ending at (x,y) and c is a constant of integration, which represents the (unknown) height of the surface at the start point. Exercise **??** asks you to show that the recovered surface does not depend on the choice of curve.

For example, we can reconstruct the surface at (u,v) by starting at $(0,0)$, summing the y-derivative along the line $x=0$ to the point $(0,v)$, and then summing the x-derivative along the line $y=v$ to the point (u,v)

$$f(u,v) = \int_0^v \frac{\partial f}{\partial y}(0,y)dy + \int_0^u \frac{\partial f}{\partial x}(x,v)dx + c$$

This is the integration path given in algorithm 1. Any other set of paths would work as well, though it is probably best to use many different paths and average, so as to spread around the error in the derivative estimates. Figure 2.14 shows the reconstruction obtained for the data of figure 2.10 and figure 2.11.

Another approach to recovering shape is to choose the function $f(x,y)$ whose partial derivatives most look like the measured partial derivatives. We explore this approach for a similar problem in section 3.5.2.

```
Obtain many images in a fixed view under different illuminants

Determine the matrix $\mathcal{V}$ from source and camera information

Create arrays for albedo, normal (3 components),
  p (measured value of $\frac{\partial f}{\partial x}$) and
  q (measured value of $\frac{\partial f}{\partial y}$)

For each point in the image array
  Stack image values into a vector $\boldsymbol{i}$
  Construct the diagonal matrix $\mathcal{I}$
  Solve $\mathcal{I}\mathcal{V}\boldsymbol{g} = \mathcal{I}\boldsymbol{i}$
    to obtain $\boldsymbol{g}$ for this point

  albedo at this point is $|\boldsymbol{g}|$
  normal at this point is $\frac{\boldsymbol{g}}{|\boldsymbol{g}|}$
  p at this point is $\frac{N_1}{N_3}$
  q at this point is $\frac{N_2}{N_3}$
end

Check:  is $(\frac{\partial p}{\partial y} - \frac{\partial q}{\partial x})^2$ small everywhere?

top left corner of height map is zero

for each pixel in the left column of height map
  height value=previous height value + corresponding q value
end

for each row
  for each element of the row except for leftmost
    height value = previous height value + corresponding p value
  end
end
```

Algorithm 2.1: *Photometric Stereo*

2.6 Interreflections: Global Shading Models

As we indicated above, local shading models can be quite misleading. In the real world, each surface patch is illuminated not only by sources, but also by other

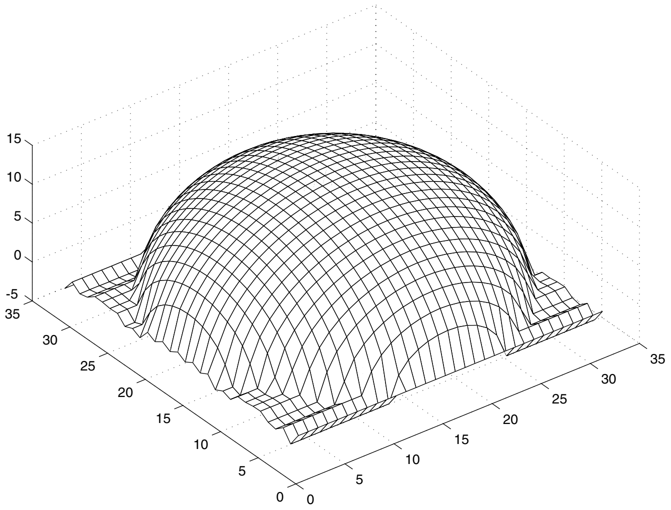


Figure 2.14. The height field obtained by integrating this normal field using the method described in the text.

surface patches. This leads to a variety of complex shading effects, which are still quite poorly understood. Unfortunately, these effects occur widely, and it is still not yet known how to simplify interreflection models without losing essential qualitative properties.

For example, Figure 2.15 shows views of the interior of two rooms. One room has black walls and contains black objects. The other has white walls, and contains white objects. Each is illuminated (approximately!) by a distant point source. Given that the intensity of the source is adjusted appropriately, the local shading model predicts that these pictures would be indistinguishable. In fact, the black room has much darker shadows and much more crisp boundaries at the creases of the polyhedra than the white room. This is because surfaces in the black room reflect less light onto other surfaces (they are darker) whereas in the white room, other surfaces are significant sources of radiation. The sections of the camera response to the radiosity (these are proportional to radiosity for diffuse surfaces) shown in the figure are hugely different qualitatively. In the black room, the radiosity is constant in patches as a local shading model would predict, whereas in the white room slow image gradients are quite common — these occur in concave corners, where object faces reflect light onto one another.

This effect also explains why a room illuminated by a point light source does not show the very sharp illumination gradients that a local shading model predicts (recall section 2.3.1). The walls and floor of the room reflect illumination back, and this tends to light up the corners, which would otherwise be dark.

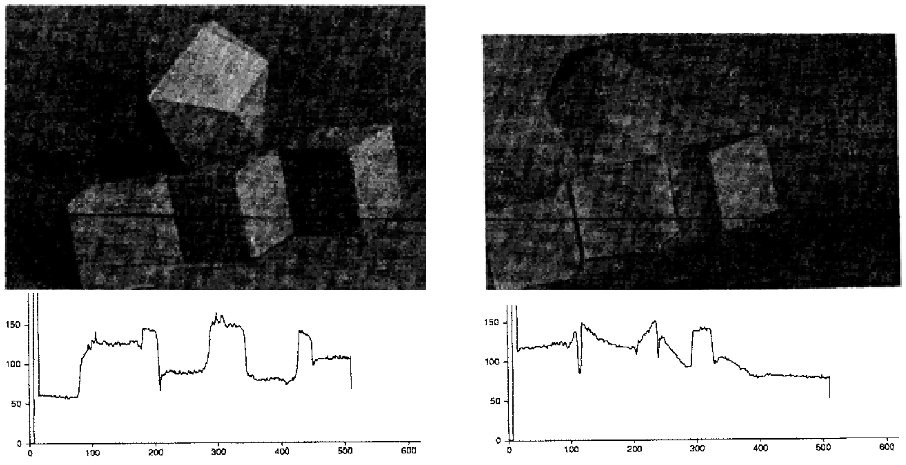


Figure 2.15. The column on the left shows data from a room with matte black walls and containing a collection of matte black polyhedral objects; that on the right shows data from a white room containing white objects. The images are qualitatively different, with darker shadows and crisper boundaries in the black room, and bright reflexes in the concave corners in the white room. The graphs show sections of the image intensity along the corresponding lines in the images. *data obtained from the paper "Mutual Illumination," by D.A. Forsyth and A.P. Zisserman, page 473 in the fervent hope that permission will be granted*

2.6.1 An Interreflection Model

It is quite well understood how to predict the radiosity on a set of diffuse surface patches. The total radiosity of a patch will be its exitance — which will be zero for all but sources — *plus* all the radiosity due to all the other patches it can see:

$$B(\boldsymbol{u}) = E(\boldsymbol{u}) + B_{incoming}(\boldsymbol{u})$$

From the point of view of our patch, there is no distinction between energy leaving another patch due to exitance and that due to reflection. This means we can take the expression for an area source, and use it to obtain an expression for $B_{incoming}(\boldsymbol{u})$. In particular, from the perspective of our patch, every other patch in the world that it can see is an area source, with exitance $B(\boldsymbol{v})$. This means that we can rework equation 2.5 to get

$$\begin{aligned} B_{incoming}(\boldsymbol{u}) &= \rho_d(\boldsymbol{u}) \int_{world} \text{visible}(\boldsymbol{u}, \boldsymbol{v}) B(\boldsymbol{v}) \frac{\cos\theta_u \cos\theta_v}{\pi d_{uv}^2} dA_{\boldsymbol{v}} && (2.6.1) \\ &= \rho_d(\boldsymbol{u}) \int_{world} \text{visible}(\boldsymbol{u}, \boldsymbol{v}) K(\boldsymbol{u}, \boldsymbol{v}) B(\boldsymbol{v}) dA_{\boldsymbol{v}} && (2.6.2) \end{aligned}$$

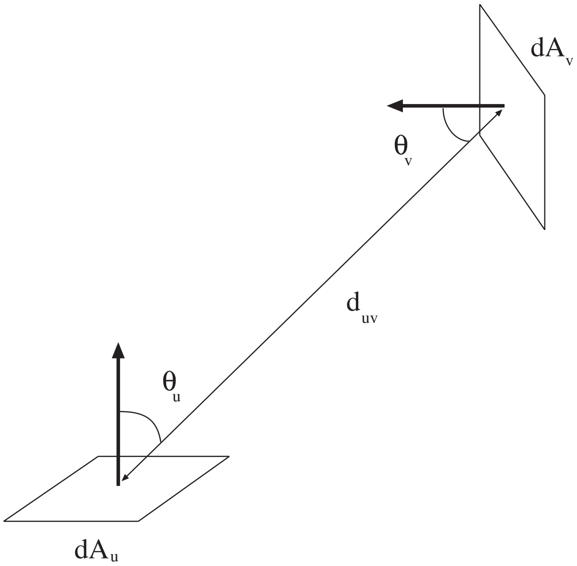


Figure 2.16. Terminology for expression derived in the text for the interreflection kernel.

where the terminology is that of Figure 2.16 and

$$\text{visible}(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} 1 \text{ if } \boldsymbol{u} \text{ can see } \boldsymbol{v}, \\ 0 \text{ if } \boldsymbol{u} \text{ can't see } \boldsymbol{v} \end{cases}$$

$\text{visible}(\boldsymbol{u}, \boldsymbol{v})K(\boldsymbol{u}, \boldsymbol{v})$ is usually referred to as the **interreflection kernel**. This means that our model is:

$$B(\boldsymbol{u}) = E(\boldsymbol{u}) + \rho_d(\boldsymbol{u}) \int_{world} \text{visible}(\boldsymbol{u}, \boldsymbol{v})K(\boldsymbol{u}, \boldsymbol{v})B(\boldsymbol{v})dA_{\boldsymbol{v}}$$

In particular, the solution appears inside the integral. Equations of this form are known as Fredholm integral equations of the second kind. This particular equation is a fairly nasty sample of the type, because the interreflection kernel is generally not continuous and may have singularities. Solutions of this equation can yield quite good models of the appearance of diffuse surfaces and the topic supports a substantial industry in the computer graphics community [].

2.6.2 Solving for Radiosity

We will sketch one approach to solving for radiosity, to illustrate the methods. Subdivide the world into small, flat patches and approximate the radiosity as being constant over each patch. This approximation is reasonable, because we could obtain a very accurate representation by working with small patches. Now we

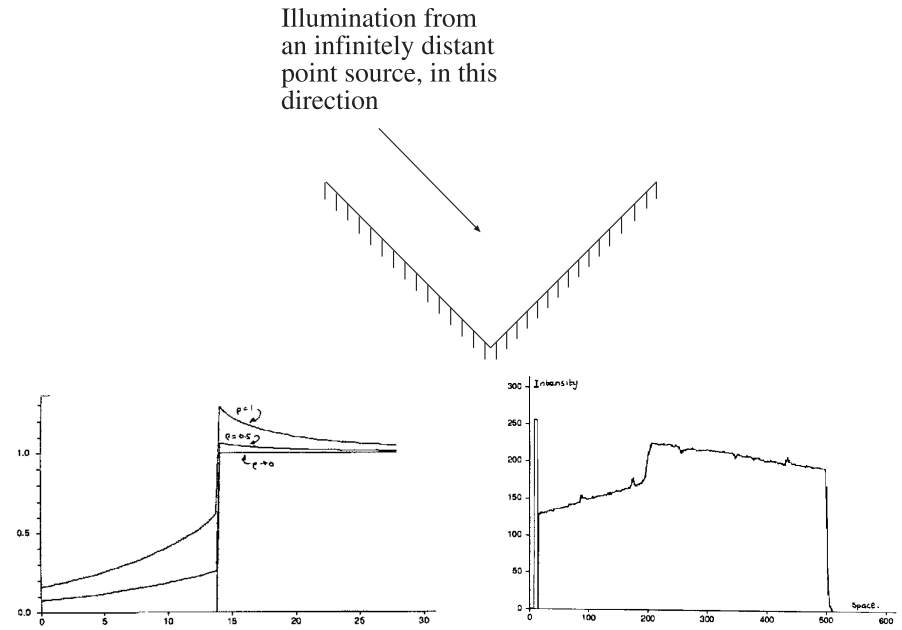


Figure 2.17. The model described in the text produces quite accurate qualitative predictions for interreflections. The top figure shows a concave right angled groove, illuminated by a point source at infinity where the source direction is parallel to the one face. On the left of the bottom row, a series of predictions of the radiosity for this configuration. These predictions have been scaled to lie on top of one another; the case $\rho \to 0$ corresponds to the local shading model. On the right, an observed image intensity for an image of this form for a corner made of white paper, showing the roof-like gradient in radiosity associated with the edge. A local shading model predicts a step. *data obtained from the paper "Mutual Illumination," by D.A. Forsyth and A.P. Zisserman, page 471 in the fervent hope that permission will be granted*

construct a vector $\boldsymbol{B}$, which contains the value of the radiosity for each patch. In particular, the i' th component of $\boldsymbol{B}$ is the radiosity of the i'th patch.

We write the incoming radiosity at the i'th patch due to radiosity on the j'th patch as $B_{j \to i}$:

$$B_{j \to i}(\boldsymbol{x}) = \rho_d(\boldsymbol{x}) \int_{\text{patch } j} \text{visible}(\boldsymbol{x}, \boldsymbol{v}) K(\boldsymbol{x}, \boldsymbol{v}) dA_{\boldsymbol{v}} B_j$$

where $\boldsymbol{x}$ is a coordinate on the i'th patch and $\boldsymbol{v}$ is a coordinate on the j'th patch. Now this expression is not a constant, and so we must average it over the i'th patch to get

$$\overline{B}_{j \to i} = \frac{1}{A_i} \int_{\text{patch } i} \rho_d(\boldsymbol{x}) \int_{\text{patch } j} \text{visible}(\boldsymbol{x}, \boldsymbol{v}) K(\boldsymbol{x}, \boldsymbol{v}) dA_{\boldsymbol{v}} dA_{\boldsymbol{x}} B_j$$

where A_i is the area of the i'th patch. If we insist that the exitance on each patch is constant, too, we obtain the model:

$$\begin{aligned} B_i &= E_i + \sum_{\text{all } j} B_{\text{average incoming at } i \text{ from } j} \\ &= E_i + \sum_{\text{all } j} K_{ij} B_j, \end{aligned}$$

where

$$K_{ij} = \frac{1}{A_i} \int_{\text{patch } i} \rho_d(\boldsymbol{x}) \int_{\text{patch } j} \text{visible}(\boldsymbol{x}, \boldsymbol{v}) K(\boldsymbol{x}, \boldsymbol{v}) dA_{\boldsymbol{v}} dA_{\boldsymbol{x}}$$

This is a system of linear equations in B_i (although an awfully big one — K_{ij} could be a million by a million matrix), and as such can in principle be solved. The tricks that are necessary to solve the system efficiently, quickly and accurately are well beyond our scope; Sillion and Puech's book is an excellent account [**?**] as is the book of Cohen [**?**].

2.6.3 The qualitative effects of interreflections

We should like to be able to extract shape information from radiosity. This is relatively easy to do with a local model (see section 2.5 for some details), but the model describes the world poorly, and very little is known about how severely this affects the resulting shape information. Extracting shape information from an interreflection model is difficult, for two reasons. Firstly, the relationship — which is governed by the interreflection kernel — between shape and radiosity is complicated. Secondly, there are almost always surfaces that are not visible, but radiate to the objects in view. These so-called "distant surfaces" mean it is hard to account for all radiation in the scene using an interreflection model, because some radiators are invisible and we may know little or nothing about them.

All this suggests that understanding qualitative, local effects of interreflection is important; armed with this understanding, we can either discount the effects of interreflection or exploit them. This topic remains largely an open research topic, but there are some things we can say.

Smoothing and Regional Properties

Firstly, interreflections have a characteristic smoothing effect. This is most obviously seen if one tries to interpret a stained glass window by looking at the pattern it casts on the floor; this pattern is almost always a set of indistinct coloured blobs. The effect is seen most easily with the crude model of Figure 6. The geometry consists of a patch with a frontal view of an infinite plane which is a unit distance away and carries a radiosity $\sin \omega x$. There is no reason to vary the distance of the patch from the plane, because interreflection problems have scale invariant solutions —

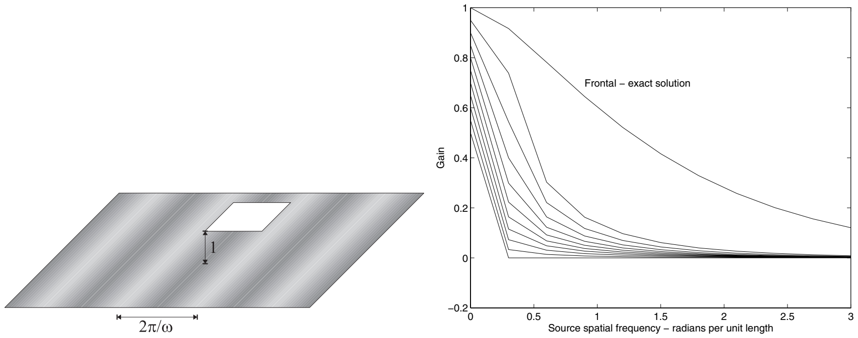


Figure 2.18. A small patch views a plane with sinusoidal radiosity of unit amplitude. This patch will have a (roughly) sinusoidal radiosity due to the effects of the plane. We refer to the amplitude of this component as the gain of the patch. The graph shows numerical estimates of the gain for patches at ten equal steps in slant angle, from 0 to $\pi/2$, as a function of spatial frequency *on the plane*. The gain falls extremely fast, meaning that large terms at high spatial frequencies must be regional effects, rather than the result of distant radiators. This is why it is hard to determine the pattern in a stained glass window by looking at the floor at foot of the window.

this means that the solution for a patch two units away can be obtained by reading our graph at 2ω. The patch is small enough that its contribution to the plane's radiosity can be ignored. If the patch is slanted by σ with respect to the plane, it carries radiosity that is nearly periodic, with spatial frequency $\omega \cos \sigma$. We refer to the amplitude of the component at this frequency as the gain of the patch, and plot the gain in Figure 6. The important property of this graph is that high spatial frequencies have a difficult time jumping the gap from the plane to the patch. This means that shading effects that have a high spatial frequency and a high amplitude generally cannot come from distant surfaces (unless they are abnormally bright).

The extremely fast fall-off in amplitude with spatial frequency of terms due to distant surfaces means that, if one observes a high amplitude term at a high spatial frequency, *it is very unlikely to have resulted from the effects of distant, passive radiators* (because these effects die away quickly). There is a convention — which we shall see in section 3.5.2 — that classifies effects in shading as due to reflectance if they are fast (“edges”) and the dynamic range is relatively low, and due to illumination otherwise. We can expand this convention. There is a mid range of spatial frequencies that are largely unaffected by mutual illumination from distant surfaces, because the gain is small. Spatial frequencies in this range cannot be “transmitted” by distant passive radiators unless these radiators have improbably high radiosity. As a result, spatial frequencies in this range can be thought of as **regional properties**, which can result only from interreflection effects within a region.

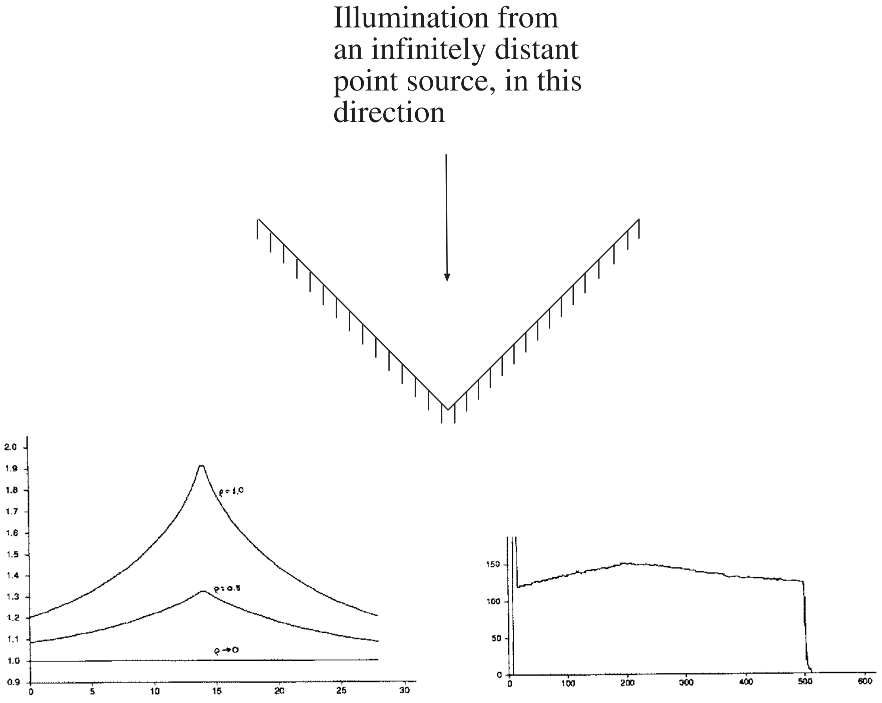


Figure 2.19. Reflexes at concave edges are a common qualitative result of interreflections. The figure on the top shows the situation here; a concave right-angled groove, illuminated by a point light source at infinity, whose source vector is along the angle bisector. The graph on the left shows the intensity predictions of an interreflection model for this configuration; the case $\rho \to 0$ is a local shading model. The graphs have been lined up for easy comparison. As the surface's albedo goes up, a roof like structure appears. The graph on the right shows an observation of this effect in an image of a real scene. *data obtained from the paper "Mutual Illumination," by D.A. Forsyth and A.P. Zisserman, page 472 in the fervent hope that permission will be granted*

The most notable regional properties are probably **reflexes**, small bright patches that appear mainly in concave regions (illustrated in Figure 2.19 and Figure 2.20). A second important effect is **colour bleeding**, where a coloured surface reflects light onto another coloured surface. This is a common effect that people tend not to notice unless they are consciously looking for it. It is quite often reproduced by painters (Figure 2.21).

2.7 Notes

Shading models are handled in a quite unsystematic way in the vision literature. The point source approximation is widely abused; you should use it with care and

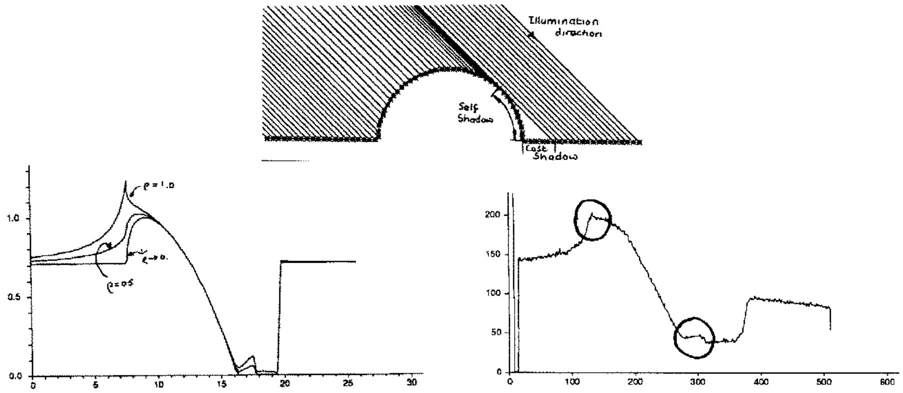


Figure 2.20. Reflexes occur quite widely; they are usually caused by a favourable view of a large reflecting surface. In the geometry shown on the left, the shadowed region of the cylindrical bump sees the plane background at a fairly favourable angle — if the background is large enough, near half the hemisphere of the patch at the base of the bump is a view of the plane. This means there will be a reflex with a large value attached to the edge of the bump, and inside the cast shadow region (which a local model predicts as black). There is another reflex on the other side, too, as the series of solutions (again, normalised for easy comparison) in the center show. On the right, an observation of this effect in a real scene. *data obtained from the paper "Mutual Illumination," by D.A. Forsyth and A.P. Zisserman, page 473 in the fervent hope that permission will be granted*

inspect others' use of it with suspicion. We believe we are the first to draw the distinction between (a) the physical effects of sources and (b) the shading model. Interreflection effects are often ignored, which causes a reflex response of hostility in one of the authors. If interreflection effects do not change the output of a method much, then it is probably all right to ignore them. Unfortunately, this line of reasoning is seldom pursued, because it is quite difficult to show that a method is stable under interreflections. This means that there is not much knowledge about the overall properties of interreflected shading other than the spatial frequency issues discussed above, which is a pity.

There are a variety of variations on photometric stereo. One interesting idea is to illuminate the surface with three lights of different colours (and in different positions) and use a colour image. For an appropriate choice of colours, this is equivalent to obtaining three images, so the measurement process is simplified.

Generally, photometric stereo is used under circumstances where the illumination is quite easily controlled, so that it is possible to ensure that there is no ambient illumination in the image. It is relatively simple to insert ambient illumination into the formulation given above; we extend the matrix $\mathcal{V}$ by attaching a column of ones. In this case, $\boldsymbol{g}(x, y)$ becomes a four dimensional vector, and the fourth component is the ambient term. However, this approach does not guarantee that the ambient term is constant over space; instead, we would have to check that this term was

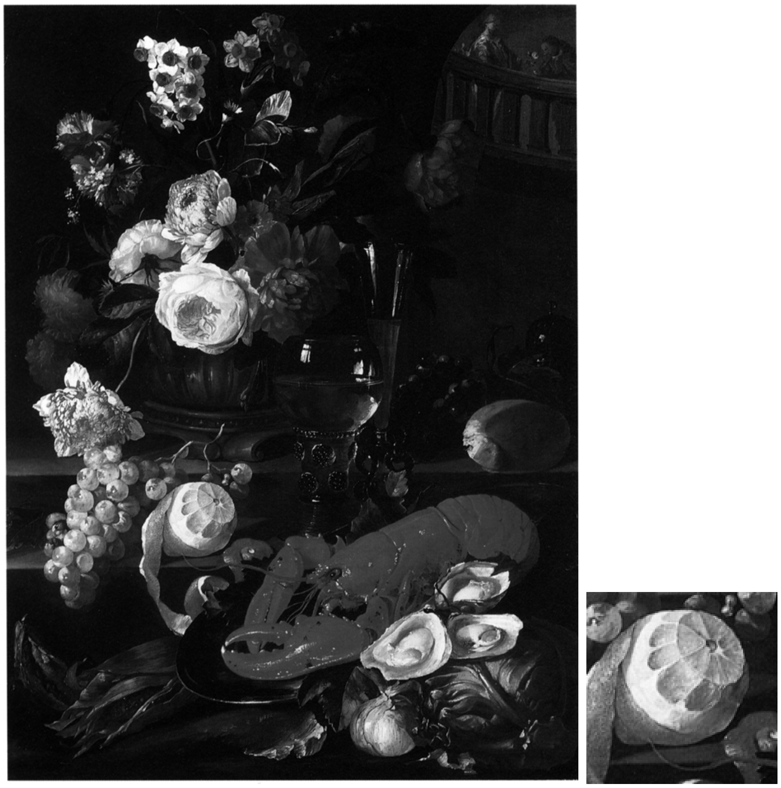


Figure 2.21. Another form of reflex occurs when colour is reflected from a surface; it is occasionally reproduced by artists. On the top, "Still Life", by A. Caproens (reproduced from Nash, "Plaisirs d'Amour", p. 95). On the bottom, a detail around the orange. Notice that the peel is reflecting orange light onto the pith, and creating an orange reflex. This effect is easily observed in real life, too.

constant, and adjust the model if it is not.

Photometric stereo depends only on adopting a local shading model. This model need not be a Lambertian surface illuminated by a distant point source. If the radiosity of the surface is a known function of the surface normal satisfying a small number of constraints, photometric stereo is still possible. This is because the intensity of a pixel in a single view determines the normal up to a one parameter family. This means that two views determine the normal. The simplest example of this case occurs for a surface of known albedo illuminated by a distant point source.

In fact, if the radiosity of the surface is a k-parameter function of the surface normal, photometric stereo is still possible. The intensity of the pixel in a single view determines the normal up to a $k+1$ parameter family, and $k+1$ views give the normal. For this approach to work, the radiosity needs to be given by a function for which our arithmetic works — if the radiosity of the surface is a constant function of the surface normal, it isn't possible to infer any constraint on the normal from the radiosity.

2.8 Assignments

2.8.1 Exercises

1. What shapes can the shadow of a sphere take, if it is cast on a plane, and the source is a point source?

2. We have a square area source and a square occluder, both parallel to a plane. The source is the same size as the occluder, and they are vertically above one another, with their centers aligned.

    - What is the shape of the umbra?
    - What is the shape of the outside boundary of the penumbra?

3. We have a square area source and a square occluder, both parallel to a plane. The edge length of the source is now twice that of the occluder, and they are vertically above one another, with their centers aligned.

    - What is the shape of the umbra?
    - What is the shape of the outside boundary of the penumbra?

4. We have a square area source and a square occluder, both parallel to a plane. The edge length of the source is now half that of the occluder, and they are vertically above one another, with their centers aligned.

    - What is the shape of the umbra?
    - What is the shape of the outside boundary of the penumbra?

5. A small sphere casts a shadow on a larger sphere. Describe the possible shadow boundaries that occur.

6. Explain why it is difficult to use shadow boundaries to infer shape, particularly if the shadow is cast onto a curved surface.

7. An infinitesimal patch views a circular area source of constant exitance frontally along the axis of symmetry of the source. Compute the radiosity of the patch, due to the source exitance $E(\boldsymbol{u})$ as a function of the area of the source and the distance between the center of the source and the patch. You may have to look the integral up in tables — if you don't, you're entitled to feel pleased with yourself — but this is one of few cases that can be done in closed form. It will be easier to look up if you transform it to get rid of the cosine terms.

8. As in Figure 6, a small patch views an infinite plane at unit distance. The patch is sufficiently small that it reflects a trivial quantity of light onto the plane. The plane has radiosity $B(x, y) = 1 + \sin ax$. The patch and the plane are parallel to one another. We will move the patch around parallel to the plane, and consider its radiosity at various points.
    - Show that if one translates the patch, its radiosity varies periodically with its position in x.
    - Fix the patches center at $(0, 0)$; determine a *closed form* expression for the radiosity of the patch at this point, as a function of a. You'll need a table of integrals for this.
9. If one looks across a large bay in the daytime, it is often hard to distinguish the mountains on the opposite side; near sunset, they are clearly visible. This phenomenon has to do with scattering of light by air — a large volume of air is actually a source. Explain what is happening. We have modelled air as a vacuum, and asserted that no energy is lost along a straight line in a vacuum. Use your explanation to give an estimate of the kind of scales over which that model is acceptable.
10. Read the book "Colour and light in nature", by Lynch and Livingstone, published by Cambridge University Press, 1995.

2.8.2 Programming Assignments

- An area source can be approximated as a grid of point sources. The weakness of this approximation is that the penumbra contains quantization errors, which can be quite offensive to the eye.
    1. Explain.
    2. Render this effect for a square source and a single occluder, casting a shadow onto an infinite plane. For a fixed geometry, you should find that as the number of point sources goes up, the quantization error goes down.
    3. This approximation has the unpleasant property that it is possible to produce arbitrarily large quantization errors with any finite grid, by changing the geometry. This is because there are configurations of source and occluder that produce very large penumbrae. Use a square source and a single occluder casting a shadow onto an infinite plane, to explain this effect.
- Make a world of black objects and another of white objects (paper, glue and spraypaint are useful here) and observe the effects of interreflections. Can you come up with a criterion that reliably tells, *from an image* which is which? (if you can, publish it; the problem looks easy, but isn't).

- (This exercise requires some knowledge of numerical analysis.) Do the numerical integrals required to reproduce Figure 6. These integrals aren't particularly easy: if one uses coordinates on the infinite plane, the size of the domain is a nuisance, and if one converts to coordinates on the view hemispher of the patch, the frequency of the radiance becomes infinite at the boundary of the hemisphere. The best way to estimate these integrals is using a Monte Carlo method on the hemisphere. You should use importance sampling, because the boundary contributes rather less to the integral than the top.

- Set up and solve the linear equations for an interreflection solution for the interior of a cube with a small square source in the center of the ceiling.

- Implement a photometric stereo system.

    1. How accurate are its measurements (i.e. how well do they compare with known shape information)? do interreflections affect the accuracy?
    2. How repeatable are its measurements (i.e. if you obtain another set of images, perhaps under different illuminants, and recover shape from those, how does the new shape compare with the old)?
    3. Compare the minimization approach to reconstruction with the integration approach; which is more accurate, or more repeatable and why? Does this difference appear in experiment?
    4. One possible way to improve the integration approach is to obtain depths by integrating over many different paths, and then average these depths (you need to be a little careful about constants here). Does this improve the accuracy or repeatability of the method?

Chapter 3

COLOUR

Colour is a rich and complex experience, usually caused by the vision system responding differently to different wavelengths of light (other causes include pressure on the eyeball and dreams). While the colour of objects seems to be a useful cue in identifying them, it is currently difficult to use.

We will first describe the physical causes of colour; we then study human colour perception, which will yield methods for describing colours; finally, we discuss how to extract information about the colour of the surfaces we are looking at from the colour of image pixels, which are affected by both surface colour and illuminant colour.

3.1 The Physics of Colour

We will extend our radiometric vocabulary to describe energy arriving in different quantities at different wavelengths and then describe typical properties of coloured surfaces and coloured light sources.

3.1.1 Radiometry for Coloured Lights: Spectral Quantities

All of the physical units we have described can be extended with the phrase "per unit wavelength" to yield **spectral units**. These allow us to describe differences in energy, in BRDF or in albedo with wavelength. We will ignore interactions where energy changes wavelength; thus, the definitions of Chapter 1 can be extended by adding the phrase "per unit wavelength," to obtain what are known as **spectral quantities**.

Spectral radiance is usually written as $L^{\lambda}(\boldsymbol{x}, \theta, \phi)$, and the radiance emitted in the range of wavelengths $[\lambda, \lambda + d\lambda]$ is $L^{\lambda}(\boldsymbol{x}, \theta, \phi)d\lambda$. Spectral radiance has units Watts per cubic meter per steradian ($Wm^{-3}sr^{-1}$ — cubic meters because of the additional factor of the wavelength). For problems where the angular distribution of the source is unimportant, **spectral exitance** is the appropriate property; spectral exitance has units Wm^{-3}.

Similarly, the **spectral BRDF** is obtained by considering the ratio of the spectral radiance in the outgoing direction to the **spectral irradiance** in the incident direction. Because the BRDF is defined by a ratio, the spectral BRDF will again have units sr^{-1}.

3.1.2 The Colour of Surfaces

The colour of coloured surfaces is a result of a large variety of mechanisms, including differential absorbtion at different wavelengths, refraction, diffraction and bulk scattering (for more details, see, for example []). Usually these effects are bundled into a macroscopic BRDF model, which is typically a Lambertian plus specular approximation; the terms are now **spectral reflectance** (sometimes abbreviated to **reflectance**) or (less commonly) **spectral albedo**. Figures 3.1 and 3.2 show examples of spectral reflectances for a number of different natural objects.

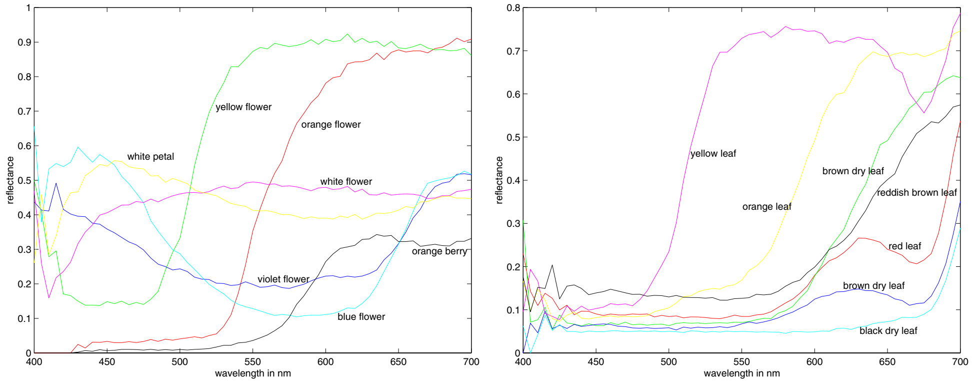


Figure 3.1. Spectral albedoes for a variety of natural surfaces, measured by Esa Koivisto, Department of Physics, University of Kuopio, Finland. On the left, albedoes for a series of different natural surfaces — a colour name is given for each. On the right, albedoes for different colours of leaf; again, a colour name is given for each. These figures were plotted from data available at `http://www.it.lut.fi/research/color/lutcs_database.html`.

The colour of the light returned to the eye is affected both by the spectral radiance (colour!) of the illuminant and by the spectral reflectance (colour!) of the surface. If we use the Lambertian plus specular model, we have:

$$E(\lambda) = \rho_{dh}(\lambda)S(\lambda) \times \text{geometric terms} + \text{specular terms}$$

where $E(\lambda)$ is the spectral radiosity of the surface, $\rho_{dh}(\lambda)$ is the spectral reflectance and $S(\lambda)$ is the spectral irradiance. The specular terms have different colours depending on the surface — i.e. we now need a **spectral specular albedo**.

Colour and Specular Reflection

Generally, metal surfaces have a specular component that is wavelength dependent — a shiny copper penny has a yellowish glint. Surfaces that do not conduct — **dielectric surfaces** — have a specular component that is independent of wavelength

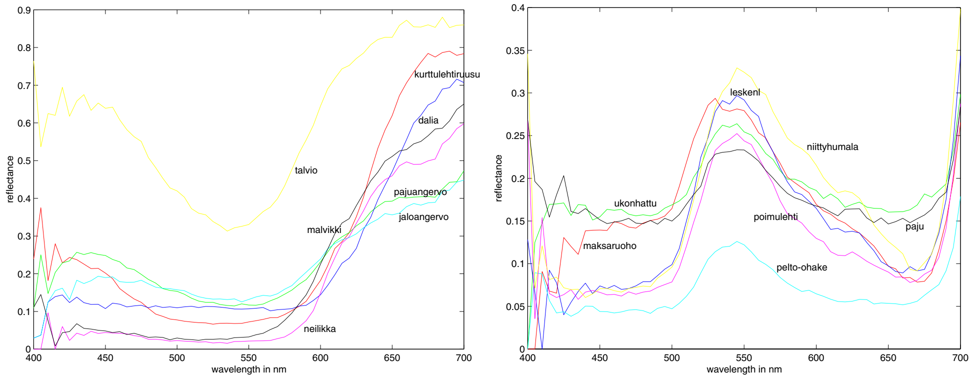


Figure 3.2. Spectral albedoes for a variety of natural surfaces, measured by Esa Koivisto, Department of Physics, University of Kuopio, Finland. On the left, albedoes for a series of different red flowers. Each is given its Finnish name. On the right, albedoes for green leaves; again, each is given its Finnish name. You should notice that these albedoes don't vary all that much. This is because there are relatively few mechanisms that give rise to colour in plants. These figures were plotted from data available at `http://www.it.lut.fi/research/color/lutcs_database.html`.

— for example, the specularities on a shiny plastic object are the colour of the light. Section 3.4 describes how these properties can be used to find specularities, and to find image regions corresponding to metal or plastic objects.

3.1.3 The Colour of Sources

Building a light source usually involve heating something until it glows. There is an idealisation of this process, which we study first. We then describe the spectral power distribution of sunlight, and discuss a number of artificial light sources.

Black Body Radiators

A body that reflects no light — usually called a **black body** — is the most efficient radiator of illumination. A heated black body emits electromagnetic radiation. It is a remarkable fact that the spectral power distribution of this radiation depends only on the temperature of the body. It is possible to build quite good black bodies (one obtains a hollow piece of metal and looks into the cavity through a tiny hole — very little of the light getting into the hole will return to the eye), so that the spectral power distribution can be measured. In particular, if we write T for the temperature of the body in Kelvins, h for Planck's constant, k for Boltzmann's constant, c for the speed of light and λ for the wavelength, we have that

$$E(\lambda) \propto \frac{1}{\lambda^5} \frac{1}{(\exp(hc/k\lambda) - 1}$$

This means that there is one parameter family of light colours corresponding to black body radiators — the parameter being the temperature — and so we can talk about the **colour temperature** of a light source. This is the temperature of the black body that looks most similar.

The Sun and the Sky

The most important natural light source is the sun. The sun is usually modelled as a distant, bright point. The colour of sunlight varies with time of day (figure 3.3) and time of year. These effects have been widely studied. Figure **??** shows one standard model of sunlight that is widely used.

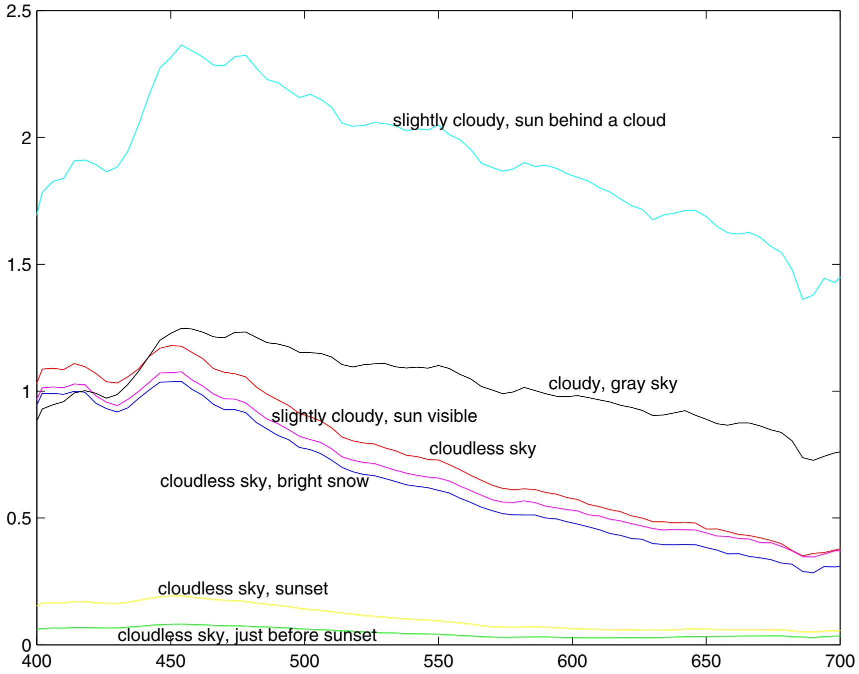


Figure 3.3. There are significant variations in the relative spectral power of sunlight measured at different times of day and under different conditions. The figure shows a series of seven different sunlight measurements, made by Jussi Parkkinen and Pertti Silfsten, of daylight illuminating a sample of barium sulphate (which gives a very high reflectance white surface). Plot from data obtainable at `http://www.it.lut.fi/research/color/lutcs_database.html`.

The sky is another important natural light source. A crude geometrical model is as a hemisphere with constant exitance. The assumption that exitance is constant is poor, however, because the sky is substantially brighter at the horizon than at the zenith. The sky is bright because light from the sun is scattered by the air.

The natural model is to consider air as emitting a constant amount of light per unit volume; this means that the sky is brighter on the horizon than at the zenith, because a viewing ray along the horizon passes through more sky.

For clear air, the intensity of radiation scattered by a unit volume depends on the fourth power of the frequency; this means that light of a long wavelength can travel very much further before being scattered than light of a short wavelength (this is known as **Rayleigh scattering**). This means that, when the sun is high in the sky, blue light is scattered out of the ray from the sun to the earth — meaning that the sun looks yellow — and can scatter from the sky into the eye — meaning that the sky looks blue. There are standard models of the spectral radiance of the sky at different times of day and latitude, too. Surprising effects occur when there are fine particles of dust in the sky (the larger particles cause very much more complex scattering effects, usually modelled rather roughly by the Mie scattering model []) — one author remembers vivid sunsets in Johannesburg caused by dust in the air from mine dumps, and there are records of blue and even green moons caused by volcanic dust in the air.

Artificial Illumination

Typical artificial light sources are commonly of a small number of types.

- An **incandescent light** contains a metal filament which is heated to a high temperature. The spectrum roughly follows the black-body law, meaning that incandescent lights in most practical cases have a reddish tinge (Figure 3.10 shows the locus of colours available from the black-body law at different temperatures).

- **Fluorescent lights** work by generating high speed electrons that strike gas within the bulb; this in turn releases ultraviolet radiation, which causes phosphors coating the inside of the bulb to fluoresce. Typically the coating consists of three or four phosphors, which fluoresce in quite narrow ranges of wavelengths. Most fluorescent bulbs generate light with a bluish tinge, but bulbs that mimic natural daylight are increasingly available (figure 3.4).

- In some bulbs, an arc is struck in an atmosphere consisting of gaseous metals and inert gases. Light is produced by electrons in metal atoms dropping from an excited state, to a lower energy state. Typical of such lamps is strong radiation at a small number of wavelengths, which correspond to particular state transitions. The most common cases are **sodium arc lamps**, and **mercury arc lamps**. Sodium arc lamps produce a yellow-orange light extremely efficiently, and are quite commonly used for freeway lighting. Mercury arc lamps produce a blue-white light, and are often used for security lighting.

Figure 3.4 shows a sample of spectra from different light bulbs.

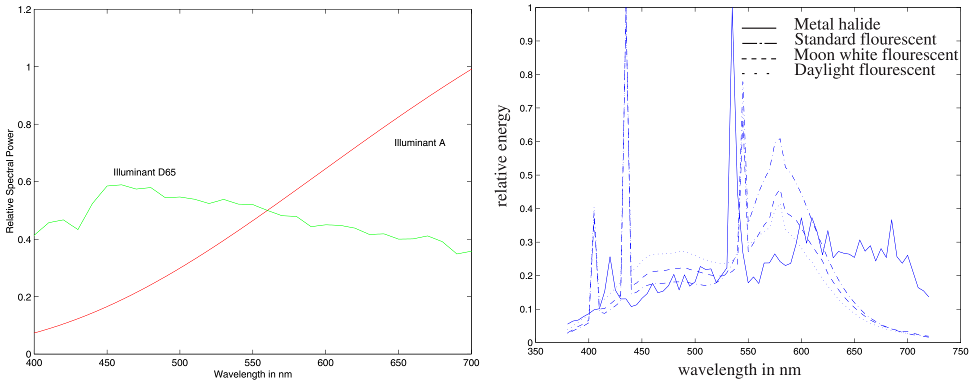


Figure 3.4. There is a variety of illuminant models; the graph shows the relative spectral power distribution of two standard CIE models, illuminant A — which models the light from a 100W Tungsten filament light bulb, with colour temperature 2800K — and illuminant D-65 — which models daylight. *Figure plotted from data available at* `http://www-cvrl.ucsd.edu/index.htm`.The relative spectral power distribution of four different lamps from the Mitsubishi Electric corp, measured by ****, data from *****. Note the bright, narrow bands that come from the flourescing phosphors in the fluorescent lamp.

3.2 Human Colour Perception

To be able to describe colours, we need to know how people respond to them. Human perception of colour is a complex function of context; illumination, memory, object identity and emotion can all play a part. The simplest question is to understand which spectral radiances produce the same response from people under simple viewing conditions (section 3.2.1). This yields a simple, linear theory of colour matching which is accurate and extremely useful for describing colours. We sketch the mechanisms underlying the transduction of colour in section 3.2.2.

3.2.1 Colour Matching

The simplest case of colour perception is obtained when only two colours are in view, on a black background. In a typical experiment a subject sees a coloured light — the **test light** — in one half of a split field. The subject can then adjust a mixture of lights in the other half to get it to match. The adjustments involve changing the intensity of some fixed number of **primaries** in the mixture. In this form, a large number of lights may be required to obtain a match, but many different adjustments may yield a match.

Write T for the test light, an equals sign for a match, the weights w_i and the primaries P_i (and so are non-negative). A match can then written in an algebraic

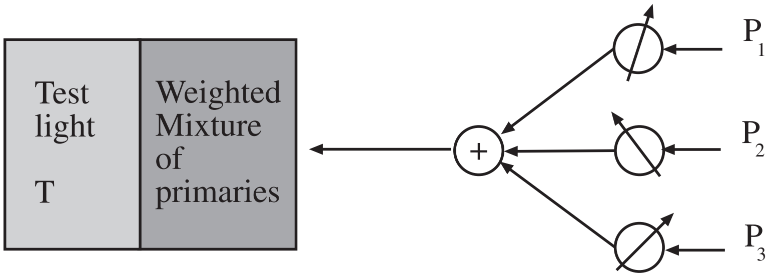


Figure 3.5. Human perception of colour can be studied by asking observers to mix coloured lights to match a test light, shown in a split field. The drawing shows the outline of such an experiment. The observer sees a test light T, and can adjust the amount of each of three primaries in a mixture that is displayed next to the test light. The observer is asked to adjust the amounts so that the mixture looks the same as the test light. The mixture of primaries can be written as $w_1 P_1 + w_2 P_2 + w_3 P_3$; if the mixture matches the test light, then we write $T = w_1 P_1 + w_2 P_2 + w_3 P_3$. It is a remarkable fact that for most people three primaries are sufficient to achieve a match for many colours, and for all colours if we allow subtractive matching (i.e. some amount of some of the primaries is mixed with the test light to achieve a match). Some people will require fewer primaries. Furthermore, most people will choose the same mixture weights to match a given test light.

form as:

$$T = w_1 P_1 + w_2 P_2 + \ldots$$

meaning that test light T matches the particular mixture of primaries given by (w_1, w_2, w_3). The situation is simplified if **subtractive matching** is allowed: in subtractive matching, the viewer can add some amount of some primaries to the test light instead of to the match. This can be written in algebraic form by allowing the weights in the expression above to be negative.

Trichromacy

It is a matter of experimental fact that for most observers only three primaries are required to match a test light. There are some caveats. Firstly, subtractive matching must be allowed, and secondly, the primaries must be independent — meaning that no mixture of two of the primaries may match a third. This phenomenon is known as the the principle of **trichromacy**. It is often explained by assuming that there are three distinct types of colour transducer in the eye; recently, evidence has emerged from genetic studies to support this view [**?**].

It is a remarkable fact that, given the same primaries and the same test light, most observers will select the same mixture of primaries to match that test light. This phenomenon is usually explained by assuming that the three distinct types of colour transducer are common to most people. Again, there is now some direct evidence from genetic studies to support this view [**?**].

Grassman's Laws

It is a matter of experimental fact that matching is (to a very accurate approximation) linear. This yields **Grassman's laws**.

Firstly, if we mix two test lights, then mixing the matches will match the result, that is, if

$$T_a = w_{a1}P_1 + w_{a2}P_2 + w_{a3}P_3$$

and

$$T_b = w_{b1}P_1 + w_{b2}P_2 + w_{b3}P_3$$

then

$$T_a + T_b = (w_{a1} + w_{b1})P_1 + (w_{a2} + w_{b2})P_2 + (w_{a3} + w_{b3})P_3$$

Secondly, if two test lights can be matched with the same set of weights, then they will match each other, that is, if

$$T_a = w_1P_1 + w_2P_2 + w_3P_3$$

and

$$T_b = w_1P_1 + w_2P_2 + w_3P_3$$

then

$$T_a = T_b$$

Finally, matching is linear: if

$$T_a = w_1P_1 + w_2P_2 + w_3P_3$$

then

$$kT_a = (kw_1)P_1 + (kw_2)P_2 + (kw_3)P_3$$

for non-negative k.

Exceptions

Given the same test light and the same set of primaries, most people will use the same set of weights to match the test light. This, trichromacy and Grassman's laws are about as true as any law covering biological systems can be. The exceptions include:

- people with aberrant colour systems as a result of genetic ill-fortune (who may be able to match everything with fewer primaries);
- people with aberrant colour systems as a result of neural ill-fortune (who may display all sorts of effects, including a complete absence of the sensation of colour);
- some elderly people (whose choice of weights will differ from the norm, because of the development of macular pigment in the eye);

- very bright lights (whose hue and saturation look different from less bright versions of the same light);
- and very dark conditions (where the mechanism of colour transduction is somewhat different than in brighter conditions).

3.2.2 Colour Receptors

Trichromacy suggests that there are profound constraints on the way colour is transduced in the eye. One hypothesis that satisfactorily explains this phenomenon is to assume that there are three distinct types of receptor in the eye that mediate colour perception. Each of these receptors turns incident light into neural signals. It is possible to reason about the sensitivity of these receptors from colour matching experiments. If two test lights that have different spectra look the same, then they must have the same effect on these receptors.

The Principle of Univariance

The **principle of univariance** states that the activity of these receptors is of one kind — i.e. they respond strongly or weakly, but do not, for example, signal the wavelength of the light falling on them. Experimental evidence can be obtained by carefully dissecting light sensitive cells and measuring their responses to light at different wavelengths, or by reasoning backward from colour matches. Univariance is a powerful idea, because it gives us a good and simple model of human reaction to coloured light: two lights will match if they produce the same receptor responses, *whatever their spectral radiances.*

Because the system of matching is linear, the receptors must be linear. Let us write p_k for the response of the k'th receptor, $\sigma_k(\lambda)$ for its sensitivity, $E(\lambda)$ for the light arriving at the receptor and Λ for the range of visible wavelengths. We can obtain the overall response of a receptor by adding up the response to each separate wavelength in the incoming spectrum so that

$$p_k = \int_\Lambda \sigma_k(\lambda)E(\lambda)d\lambda$$

Rods and Cones

Anatomical investigation of the retina shows two types of cell that are sensitive to light, differentiated by their shape. The light sensitive region of a **cone** has a roughly conical shape, whereas that in a **rod** is roughly cylindrical. Cones largely dominate colour vision and completely dominate the fovea. Cones are somewhat less sensitive to light than rods are, meaning that in low light, colour vision is poor and it is impossible to read (one doesn't have sufficient spatial precision, because the fovea isn't working).

Studies of the genetics of colour vision support the idea that there are three types of cone, differentiated by their sensitivity (in the large; there is some evidence

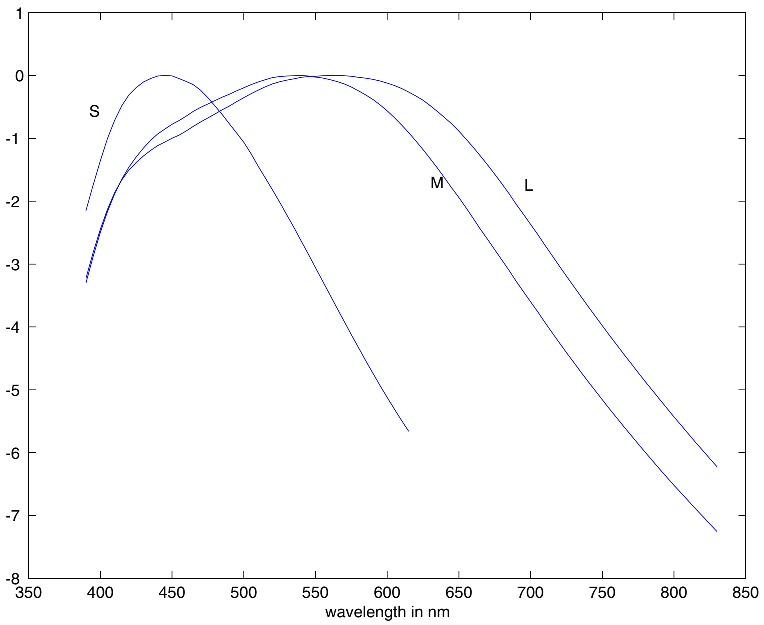


Figure 3.6. There are three types of colour receptor in the human eye, usually called cones. These receptors respond to all photons in the same way, but in different amounts. The figure shows the log of the relative spectral sensitivities of the three kinds of colour receptor in the human eye. The first two receptors —sometimes called the red and green cones respectively, but more properly named the long and medium wavelength receptors — have peak sensitivities at quite similar wavelengths. The third receptor has a very different peak sensitivity. The response of a receptor to incoming light can be obtained by summing the product of the sensitivity and the spectral radiance of the light, over all wavelengths. *Figures plotted from data available at* `http://www-cvrl.ucsd.edu/index.htm`.

that there are slight differences from person to person within each type). The sensitivities of the three different kinds of receptor to different wavelengths can be obtained by comparing colour matching data for normal observers with colour matching data for observers lacking one type of cone. Sensitivities obtained in this fashion are shown in Figure 3.6. The three types of cone are properly called **S cones**, **M cones** and **L cones** (for their peak sensitivity being to short, medium and long wavelength light respectively). They are occasionally called blue, green and red cones; this is bad practice, because the sensation of red is definitely not caused by the stimulation of red cones, etc.

3.3 Representing Colour

Describing colours accurately is a matter of great commercial importance. Many products are closely associated with very specific colours — for example, the golden arches; the colour of various popular computers; the colour of photographic film boxes — and manufacturers are willing to go to a great deal of trouble to ensure that different batches have the same colour. This requires a standard system for talking about colour. Simple names are insufficient, because relatively few people know many colour names, and most people are willing to associate a large variety of colours with a given name.

Colour matching data yields simple and and highly effective linear colour spaces (section 3.3.1). Specific applications may require colour spaces that emphasize particular properties (section 3.3.2) or uniform colour spaces, which capture the significance of colour differences (section 3.3.2).

3.3.1 Linear Colour Spaces

There is a natural mechanism for representing colour: first, agree on a standard set of primaries, and then describe any coloured light by the three values of the weights that people would use to match the light using those primaries. In principle, this is easy to use — to describe a colour, we set up and perform the matching experiment and transmit the match weights. Of course, this approach extends to give a representation for surface colours as well if we use a standard light for illuminating the surface (and if the surfaces are equally clean, etc.).

Performing a matching experiment each time we wish to describe a colour can be practical. For example, this is the technique used by paint stores; you take in a flake of paint, and they'll mix paint, adjusting the mixture until a colour match is obtained. Paint stores do this because complicated scattering effects within paints mean that predicting the colour of a mixture can be quite difficult. However, Grassman's laws mean that mixtures of coloured lights — at least those seen in a simple display — mix *linearly*, which means that a much simpler procedure is available.

Colour Matching Functions

When colours mix linearly, we can construct a simple algorithm to determine which weights would be used to match a source of some known spectral radiance, given a fixed set of primaries. The spectral radiance of the source can be thought of as a weighted sum of single wavelength sources. Because colour matching is linear, the combination of primaries that matches a weighted sum of single wavelength sources is obtained by matching the primaries to each of the single wavelength sources, and then adding up these match weights.

If we have a record of the weight of each primary required to match a single-wavelength source — a set of **colour matching functions** — we can obtain the weights used to match an arbitrary spectral radiance. The colour matching functions — which we shall write as $f_1(\lambda)$, $f_2(\lambda)$ and $f_3(\lambda)$ — can be obtained from a set of primaries P_1, P_2 and $P_3$by experiment. Essentially, we tune the weight of each primary to match a unit radiance source at every wavelength. We then obtain a set of weights, one for each wavelength, for matching a unit radiance source $U(\lambda)$. We can write this process as

$$U(\lambda) = f_1(\lambda)P_1 + f_2(\lambda)P_2 + f_3(\lambda)P_3$$

i.e. at each wavelength λ, $f_1(\lambda)$, $f_2(\lambda)$ and $f_3(\lambda)$ give the weights required to match a unit radiance source at that wavelength.

The source — which we shall write $S(\lambda)$ — is a sum of a vast number of single wavelength sources, each with a different intensity. We now match the primaries to each of the single wavelength sources, and then add up these match weights, obtaining

$$\begin{aligned} S(\lambda) &= w_1 P_1 + w_2 P_2 + w_3 P_3 \\ &= \left\{ \int_\Lambda f_1(\lambda) S(\lambda) d\lambda \right\} P_1 + \left\{ \int_\Lambda f_2(\lambda) S(\lambda) d\lambda \right\} P_2 + \left\{ \int_\Lambda f_3(\lambda) S(\lambda) d\lambda \right\} P_3 \end{aligned}$$

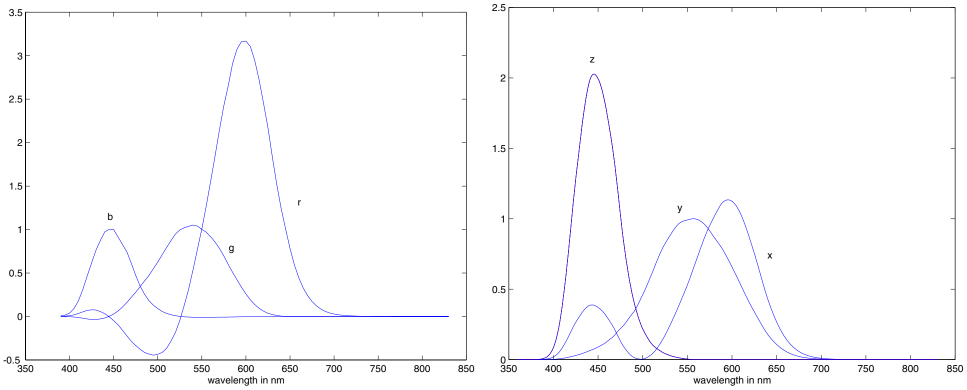


Figure 3.7. On the left, colour matching functions for the primaries for the RGB system. The negative values mean that subtractive matching is required to match lights at that wavelength with the RGB primaries. On the right, colour matching functions for the CIE X, Y and Z primaries; the colourmatching functions are everywhere positive, but the primaries are not real. Figures plotted from data available at `http://www-cvrl.ucsd.edu/index.htm`.

General Issues for Linear Colour Spaces

Linear colour naming systems can be obtained by specifying primaries — which imply colour matching functions – or by specifying colour matching functions — which imply primaries. It is an inconvenient fact of life that, if the primaries are real lights, at least one of the colour matching functions will be negative for some wavelengths. This is not a violation of natural law — it just implies that subtractive matching is required to match some lights, whatever set of primaries is used. It is a nuisance though.

One way to avoid this problem is to specify colour matching functions that are everywhere positive (which guarantees that the primaries are imaginary, because for some wavelengths their spectral radiance will be negative).

Although this looks like a problem — how would one create a real colour with imaginary primaries? — it isn't, because colour naming systems are hardly ever used that way. Usually, we would simply compare weights to tell whether colours are similar or not, and for that purpose it is enough to know the colour matching functions. A variety of different systems have been standardised by the CIE (the *commission international d'éclairage*, which exists to make standards on such things).

The CIE XYZ Colour Space

The **CIE XYZ colour space** is one quite popular standard. The colour matching functions were chosen to be everywhere positive, so that the coordinates of any real light are always positive. It is not possible to obtain CIE X, Y, or Z primaries because for some wavelengths the value of their spectral radiance is negative. However, given colour matching functions alone, one can specify the XYZ coordinates of a colour and hence describe it.

Linear colour spaces allow a number of useful graphical constructions which are more difficult to draw in three-dimensions than in two, so it is common to intersect the XYZ space with the plane $X + Y + Z = 1$ (as shown in Figure 3.8) and draw the resulting figure, using coordinates

$$(x, y) = (\frac{X}{X + Y + Z}, \frac{Y}{X + Y + Z})$$

This space is shown in Figure 3.10. CIE xy is widely used in vision and graphics textbooks and in some applications, but is usually regarded by professional colorimetrists as out of date.

The RGB Colour Spaces

Colour spaces are normally invented for practical reasons, and so a wide variety exist. The **RGB colour space** is a linear colour space that formally uses single wavelength primaries (645.16 nm for R, 526.32nm for G and 444.44nm for B — see Figure 3.7). Informally, RGB uses whatever phosphors a monitor has as primaries. Available colours are usually represented as a unit cube — usually called the **RGB cube** — whose edges represent the R, G, and B weights. The cube is drawn in figure **??**; remember, since the weights are the weights associated with primary lights, red and green mix to give yellow.

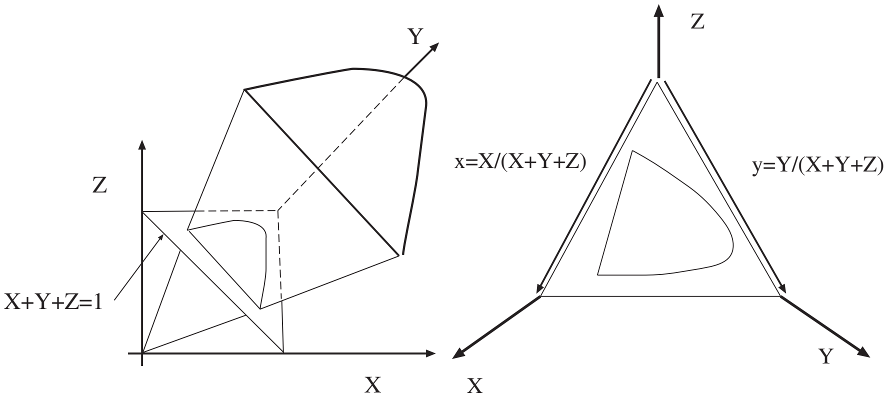


Figure 3.8. The volume of all visible colours in CIE XYZ coordinate space is a cone whose vertex is at the origin. Usually, it is easier to suppress the brightness of a colour — which we can do because to a good approximation perception of colour is linear — and we do this by intersecting the cone with the plane $X + Y + Z = 1$ to get the CIE xy space shown in figure 3.10

CMY and Black

Intuition from one's finger-painting days suggests that the primary colours should be red, yellow and blue, and that red and green mix to make yellow. The reason this intuition doesn't apply to monitors is that it is about pigments — which mix subtractively — rather than about lights. Pigments remove colour from incident light which is reflected from paper. Thus, red ink is really a dye that absorbs green and blue light — incident red light passes through this dye and is reflected from the paper.

Colour spaces for this kind of subtractive matching can be quite complicated. In the simplest case, mixing is linear (or reasonably close to linear) and the **CMY space** applies. In this space, there are three primaries: **cyan** (a blue-green colour); **magenta** (a purplish colour) and **yellow**. These primaries should be thought of as subtracting a light primary from white light; cyan is $W - R$ (white-red); magenta is $W - G$ (white-green) and yellow is $W - B$ (white-blue). Now the appearance of mixtures may be evaluated by reference to the RGB colour space. For example cyan and magenta mixed give

$$(W - R) + (W - G) = R + G + B - R - G = B$$

that is, blue. Notice that $W + W = W$ because we assume that ink cannot cause paper to reflect more light than it does when uninked. Practical printing devices use at least four inks (cyan, magenta, yellow and black), because: mixing colour

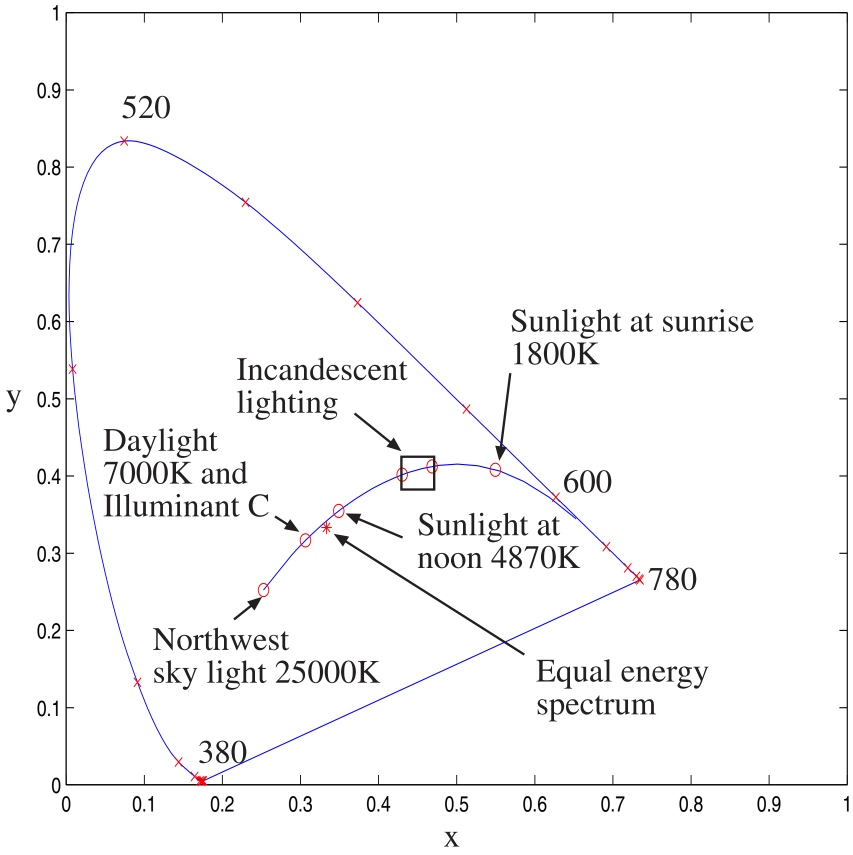


Figure 3.9. The figure shows a constant brightness section of the standard 19** standard CIE xy colour space. This space has two coordinate axes. The curved boundary of the figure is often known as the spectral locus — it represents the colours experienced when lights of a single wavelength are viewed. The figure shows a locus of colours due to black-body radiators at different temperatures, and a locus of different sky colours. Near the center of the diagram is the neutral point, the colour whose weights are equal for all three primaries. CIE selected the primaries so that this light appears achromatic. Generally, colours that lie further away from the neutral point are more saturated — the difference between deep red and pale pink — and hue — the difference between green and red — as one moves around the neutral point. (Taken in the fervent hope of receiving permission from Lamb and Bourriau, Colour Art and Science, p. 88)

inks leads to a poor black; it is difficult to ensure good enough registration between the three colour inks to avoid coloured haloes around text; and colour inks tend to be more expensive than black inks. Getting really good results from a colour printing process is still difficult: different inks have significantly different spectral

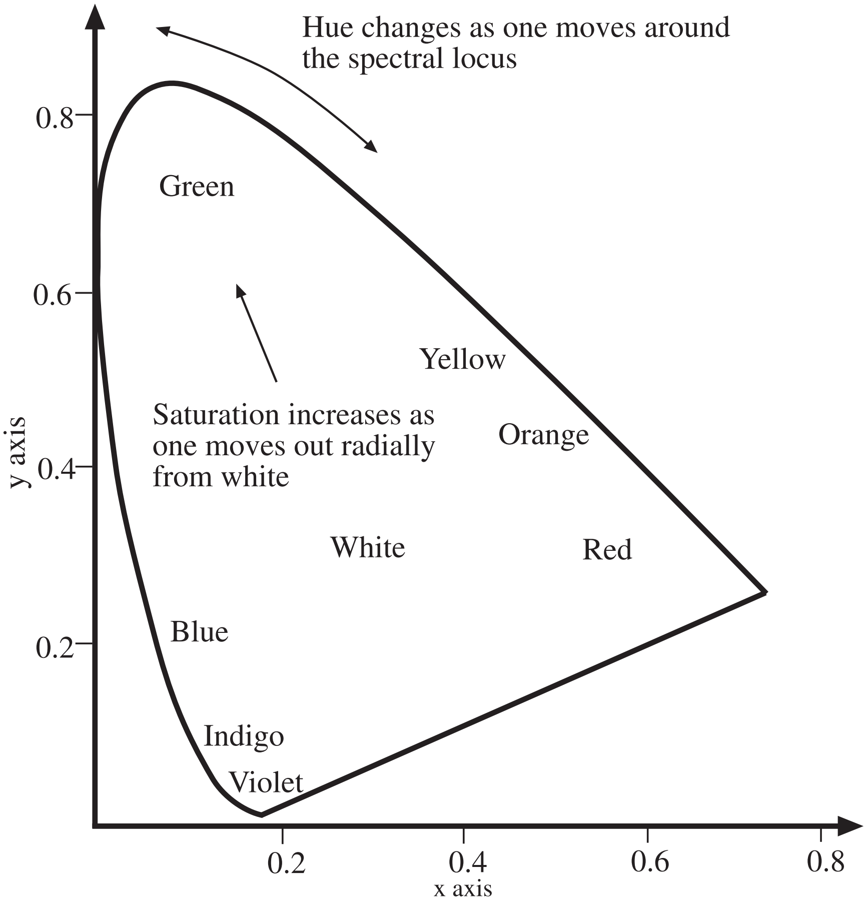

Hue changes as one moves around the spectral locus
0.8
0.6
0.4
0.2
y axis
Green
Yellow
Saturation increases as one moves out radially from white
Orange
White
Red
Blue
Indigo
Violet
0.2
0.4
0.6
0.8
x axis

Figure 3.10. The figure shows a constant brightness section of the standard 19** standard CIE xy colour space, with colour names marked on the diagram. Generally, colours that lie further away from the neutral point are more saturated — the difference between deep red and pale pink — and hue — the difference between green and red — as one moves around the neutral point. (Taken in the fervent hope of receiving permission from Lamb and Bourriau, Colour Art and Science, p. 88)

properties; different papers have different spectral properties, too; and inks can mix non-linearly.

3.3.2 Non-linear Colour Spaces

The coordinates of a colour in a linear space may not necessarily encode properties that are common in language or are important in applications. Useful colour terms include: **hue** — the property of a colour that varies in passing from red to green;

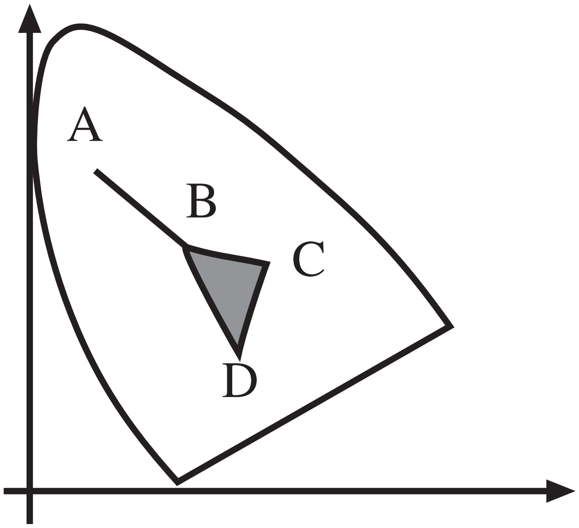


Figure 3.11. The linear model of the colour system allows a variety of useful constructions. If we have two lights whose CIE coordinates are $\boldsymbol{A}$ and $\boldsymbol{B}$ all the colours that can be obtained from non-negative mixtures of these lights are represented by the line segment joining $\boldsymbol{A}$ and $\boldsymbol{B}$. In turn, given $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$, the colours that can by obtained by mixing them lie in the triangle formed by the three points. This is important in the design of monitors — each monitor has only three phosphors, and the more saturated the colour of each phosphor the bigger the set of colours that can be displayed. This also explains why the same colours can look quite different on different monitors. The curvature of the spectral locus gives the reason that no set of three real primaries can display all colours without subtractive matching.

saturation — the property of a colour that varies in passing from red to pink; and **brightness** (sometimes called **lightness** or **value**) — the property that varies in passing from black to white. For example, if we are interested in checking whether a colour lies in a particular range of reds, we might wish to encode the hue of the colour directly.

Another difficulty with linear colour spaces is that the individual coordinates do not capture human intuitions about the topology of colours; it is a common intuition that hues form a circle, in the sense that hue changes from red, through orange to yellow and then green and from there to cyan, blue, purple and then red again. Another way to think of this is to think of local hue relations: red is next to purple and orange; orange is next to red and yellow; yellow is next to orange and green; green is next to yellow and cyan; cyan is next to green and blue; blue is next to cyan and purple; and purple is next to blue and red. Each of these local relations works, and globally they can be modelled by laying hues out in a circle. This means that no individual coordinate of a linear colour space can model hue, because that coordinate has a maximum value which is far away from the minimum value.

Hue, Saturation and Value

A standard method for dealing with this problem is to construct a colour space that reflects these relations by applying a non-linear transformation to the RGB space. There are many such spaces. One, called **HSV space** (for hue, saturation and value) is obtained by looking down the center axis of the RGB cube. Because RGB is a linear space, brightness — called value in HSV — varies with scale out from the origin, and we can "flatten" the RGB cube to get a 2D space of constant value, and for neatness deform it to be a hexagon. This gets the structure shown in figure 3.12, where hue is given by an angle that changes as one goes round the neutral point and saturation changes as one moves away from the neutral point.

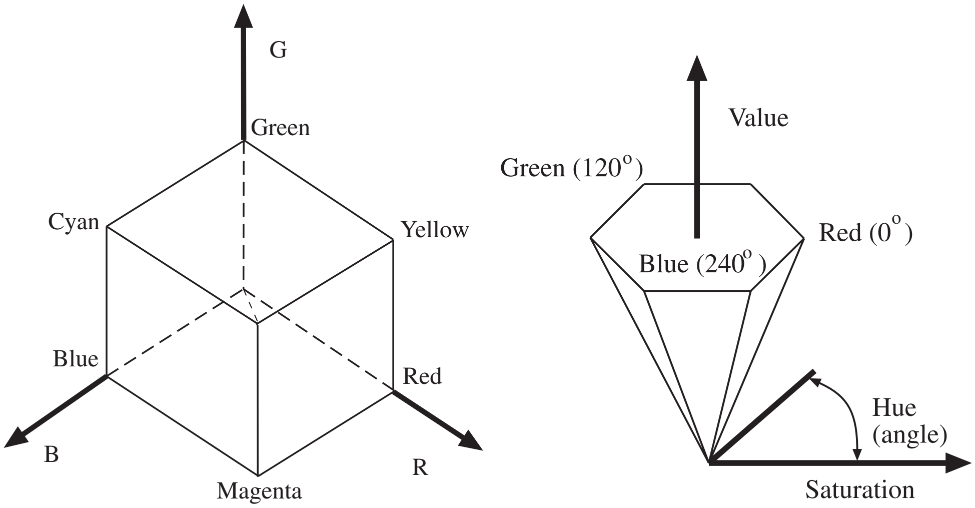


Figure 3.12. On the left, we see the RGB cube; this is the space of all colours that can be obtained by combining three primaries (R, G, and B — usually defined by the colour response of a monitor) with weights between zero and one. It is common to view this cube along its neutral axis — the axis from the origin to the point (1, 1, 1) — to see a hexagon, shown in the middle. This hexagon codes hue (the property that changes as a colour is changed from green to red) as an angle, which is intuitively satisfying. On the right, we see a cone obtained from this cross-section, where the distance along a generator of the cone gives the value (or brightness) of the colour, angle around the cone gives the hue and distance out gives the saturation of the colour.

There are a variety of other possible changes of coordinate from between linear colour spaces, or from linear to non-linear colour spaces (Fairchild's book [] is a good reference). There is no obvious advantage to using one set of coordinates over another (particularly if the difference between coordinate systems is just a one-one transformation) unless one is concerned with coding and bit-rates, etc. or with perceptual uniformity.

Uniform Colour Spaces

Usually, one cannot reproduce colours exactly. This means it is important to know whether a colour difference would be noticeable to a human viewer; it is generally useful to be able to compare the significance of small colour differences[1].

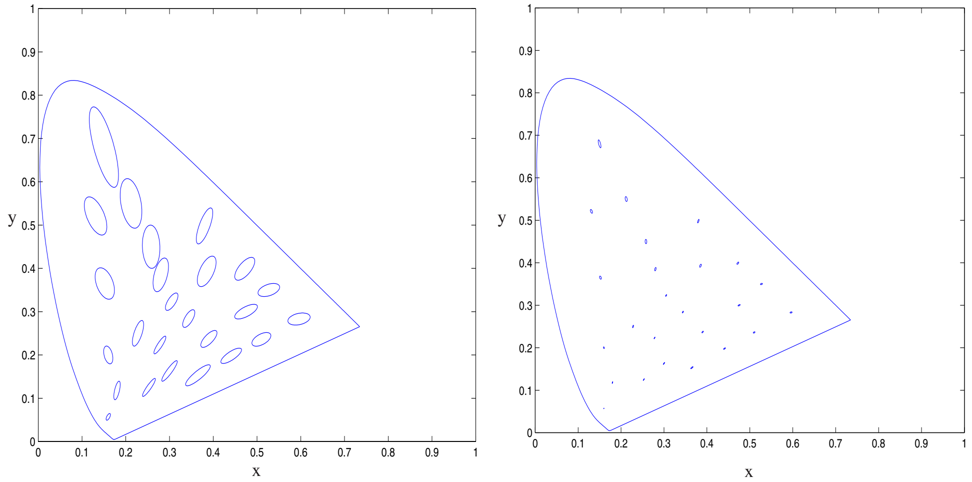


Figure 3.13. This figure shows variations in colour matches on a CIE x, y space. At the center of the ellipse is the colour of a test light; the size of the ellipse represents the scatter of lights that the human observers tested would match to the test colour; the boundary shows where the just noticeable difference is. The ellipses in the figure on the **left** have been magnified 10x for clarity, and on the **right** they are plotted to scale. The ellipses are known as MacAdam ellipses, after their inventor. Notice that the ellipses at the top are larger than those at the bottom of the figure, and that they rotate as they move up. This means that the magnitude of the difference in x, y coordinates is a poor guide to the difference in colour. Ellipses plotted using data from Macadam's paper of 194*

Just noticeable differences can be obtained by modifying a colour shown to an observer until they can only just tell it has changed in a comparison with the original colour. When these differences are plotted on a colour space, they form the boundary of a region of colours that are indistinguishable from the original colours. Usually, ellipses are fitted to the just noticeable differences. It turns out that in CIE xy space these ellipses depend quite strongly on where in the space the difference occurs, as the Macadam ellipses in Figure 3.13 illustrate.

---

[1] It is usually dangerous to try and compare large colour differences; consider trying to answer the question "is the blue patch more different from the yellow patch than the red patch is from the green patch?"

This means that the size of a difference in (x, y) coordinates, given by $(\Delta x)^2 + (\Delta y)^2)$, is a poor indicator of the significance of a difference in colour (if it was a good indicator, the ellipses representing indistinguishable colours would be circles). A **uniform colour space** is one in which the distance in coordinate space is a fair guide to the significance of the difference between two colours — in such a space, if the distance in coordinate space was below some threshold, then a human observer would not be able to tell the colours apart.

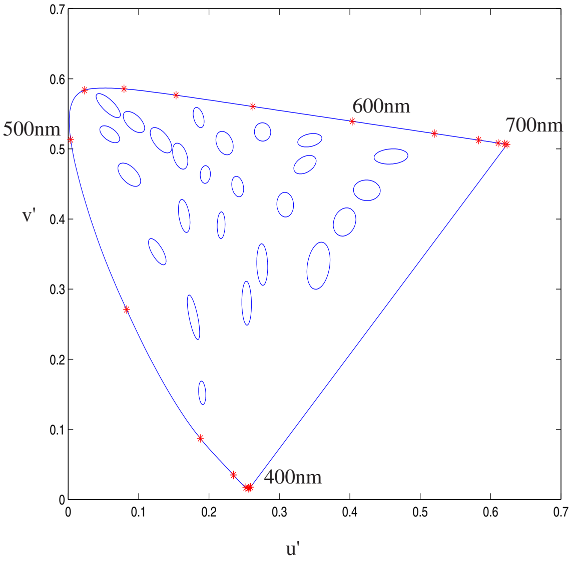


Figure 3.14. This figure shows the CIE 1976 u', v' space, which is obtained by a projective transformation of CIE x, y space. The intention is to make the MacAdam ellipses uniformly circles — this would yield a uniform colour space. A variety of non-linear transforms can be used to make the space more uniform (see [**?**] for details)

A more uniform space can be obtained from CIE XYZ by using a projective transformation to skew the ellipses; this yields the **CIE u'v' space**, illustrated in Figure 3.14. The coordinates are:

$$(u', v') = (\frac{4X}{X + 15Y + 3Z}, \frac{9Y}{X + 15Y + 3Z})$$

Generally, the distance between coordinates in u', v' space is a fair indicator of the significance of the difference between two colours. Of course, this omits differences in brightness. **CIE LAB** is now almost universally the most popular uniform colour space. Coordinates of a colour in LAB are obtained as a non-linear mapping of the XYZ coordinates:

$$
\begin{aligned}
L^* &= 116\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - 16 \\
a^* &= 500\left[\left(\frac{X}{X_n}\right)^{\frac{1}{3}} - \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}}\right] \\
b^* &= 200\left[\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_n}\right)^{\frac{1}{3}}\right]
\end{aligned}
$$

(here X_n, Y_n and Z_n are the X, Y, and Z coordinates of a reference white patch). The reason to care about the LAB space is that it is substantially uniform. In some problems, it is important to understand how different two colours will look *to a human observer*, and differences in LAB coordinates give a good guide.

3.3.3 Spatial and Temporal Effects

Predicting the appearance of complex displays of colour — i.e. a stimulus that is more interesting than a pair of lights — is difficult. If the visual system has been exposed to a particular illuminant for some time, this causes the colour system to adapt, a process known as **chromatic adaptation**. Adaptation causes the colour diagram to skew, in the sense that two observers, adapted to different illuminants, can report that spectral radiosities with quite different chromaticities have the same colour. Adaptation can be caused by surface patches in view. Other mechanisms that are significant are **assimilation** — where surrounding colours cause the colour reported for a surface patch to move towards the colour of the surrounding patch — and **contrast** — where surrounding colours cause the colour reported for a surface patch to move away from the colour of the surrounding patch. These effects appear to be related to coding issues within the optic nerve, and colour constancy (section 3.5).

3.4 Application: Finding Specularities

Specularities can have quite strong effects on the appearance of an object. Typically, they appear as small, bright patches, often called highlights. Highlights have a substantial effect on human perception of a surface properties; the addition of small, highlight-like patches to a figure makes the object depicted look glossy or shiny. Specularities are often sufficiently bright to saturate the camera, so that the colour can be hard to measure. However, because the appearance of a specularity is quite strongly constrained, there are a number of effective schemes for marking them, and the results can be used as a shape cue.

The dynamic range of practically available albedoes is relatively small. Surfaces with very high or very low albedo are difficult to make. Uniform illumination is common, too, and most cameras are reasonably close to linear within their operating range. This means that very bright patches cannot be due to diffuse reflection; they must be either sources (of one form or another — perhaps a stained glass window with the light behind it) or specularities [**?**]. Furthermore, specularities tend to be small. Thus, looking for small very bright patches can be an effective way of finding specularities [].

In colour images, specularities produce quite characteristic effects if they occur on **dielectric** materials (those that do not conduct electricity). This link to conductivity occurs because electric fields cannot penetrate conductors (the electrons inside just move around to cancel the field), so that light striking a metal surface can be either absorbed or specularly reflected. Dull metal surfaces look dull because of surface roughness effects and shiny metal surfaces have shiny patches that have a characteristic colour because the conductor absorbs energy in different amounts at different wavelengths. However, light striking a dielectric surface can penetrate it. Many dielectric surfaces can be modelled as a clear matrix with randomly embedded pigments; this is a particularly good model for plastics and for some paints. In this model, there are two components of reflection that correspond to our specular and diffuse notions: **body reflection**, which comes from light penetrating the matrix, striking various pigments and then leaving; and **surface reflection**, which comes from light specularly reflected from the surface. Assuming the pigment is randomly distributed (and small, and not on the surface, etc.) and the matrix is reasonable, we have that the body reflection component will behave like a diffuse component with a spectral albedo that depends on the pigment and the surface component will be independent of wavelength.

Assume we are looking at a single object dielectric object with a single colour. We expect that the interreflection term can be ignored, and our model of camera pixel brightnesses becomes

$$\boldsymbol{p}(\boldsymbol{x}) = g_d(\boldsymbol{x})\boldsymbol{d} + g_s(\boldsymbol{x})\boldsymbol{s}$$

where $\boldsymbol{s}$ is the colour of the source and $\boldsymbol{d}$ is the colour of the diffuse reflected light, $g_d(\boldsymbol{x})$ is a geometric term that depends on the orientation of the surface and $g_s(\boldsymbol{x})$ is a term that gives the extent of the specular reflection. If the object is curved, then $g_s(\boldsymbol{x})$ is small over much of the surface, and large only around specularities; and $g_d(\boldsymbol{x})$ varies more slowly with the orientation of the surface. We now map the colours produced by this surface in receptor response space, and look at the structures that appear there (Figure 3.15).

The term $g_d(\boldsymbol{x})\boldsymbol{d}$ will produce a line that should extend to pass through the origin, because it represents the same vector of receptor responses multiplied by a constant that varies over space. If there is a specularity, then we expect to see a second line, due to $g_s(\boldsymbol{x})\boldsymbol{s}$. This will not, in general, pass through the origin (because of the diffuse term). This is a line, rather than a planar region, because $g_s(\boldsymbol{x})$ is large over only a very small range of surface normals, and we expect that, because the surface is curved, this corresponds to a small region of surface. The term $g_d(\boldsymbol{x})$ should be approximately constant in this region. We expect a line,

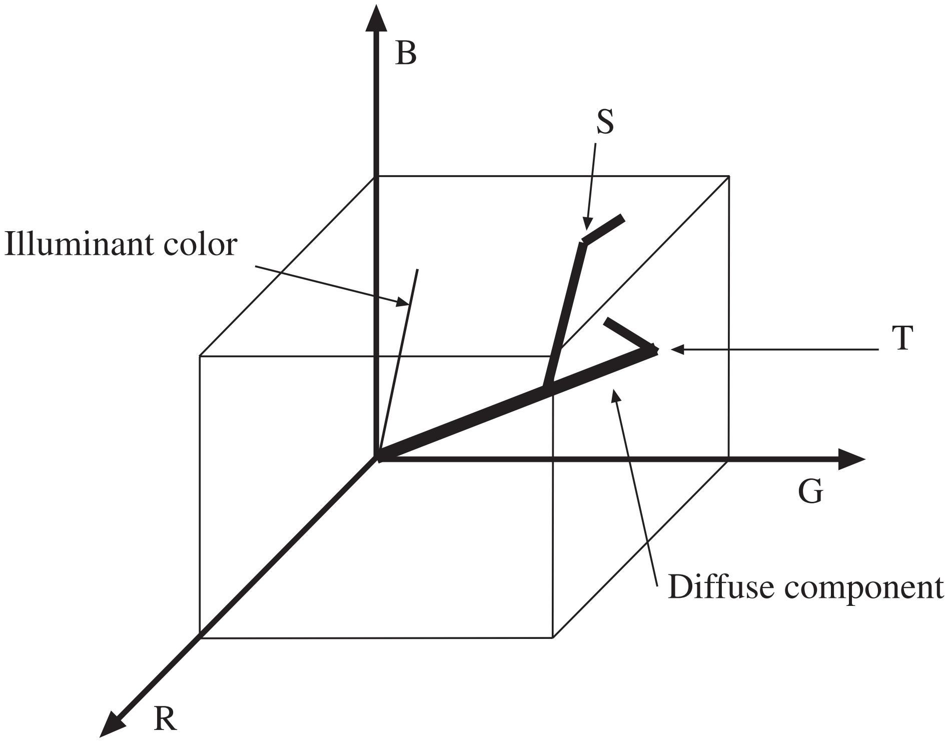


Figure 3.15. Assume we have a picture of a single uniformly coloured surface. Our model of reflected light should lead to a gamut that looks like the drawing. We are assuming that reflected light consists of the diffuse term plus a specular term, and the specular term is the colour of the light source. Most points on the surface do not have a significant specular term, and instead are brighter or darker versions of the same diffuse surface colour. At some points, the specular term is large, and this leads to a "dog-leg" in the gamut, caused by adding the diffuse term to the source term. If the diffuse reflection is very bright, one or another colour channel might saturate (point T); similarly, if the specular reflection is very bright one or another colour channel might saturate (point "S").

rather than an isolated pixel value, because we expect surfaces to have (possibly narrow) specular lobes, meaning that the specular coefficient has a range of values. This second line may collide with a face of the colour cube and get clipped.

The resulting dog-leg pattern leads pretty much immediately to a specularity marking algorithm — find the pattern, and then find the specular line. All the pixels on this line are specular pixels, and the specular and diffuse components can be estimated easily. For the approach to work effectively, we need to be confident that only one object is represented in the collection of pixels. This is helped by using local image windows as illustrated by Figure 3.16. The observations underlying the method hold if the surface is not monochrome — a coffee mug with a picture on it, for example — but finding the resulting structures in the colour space now becomes something of a nuisance, and to our knowledge has not been demonstrated.

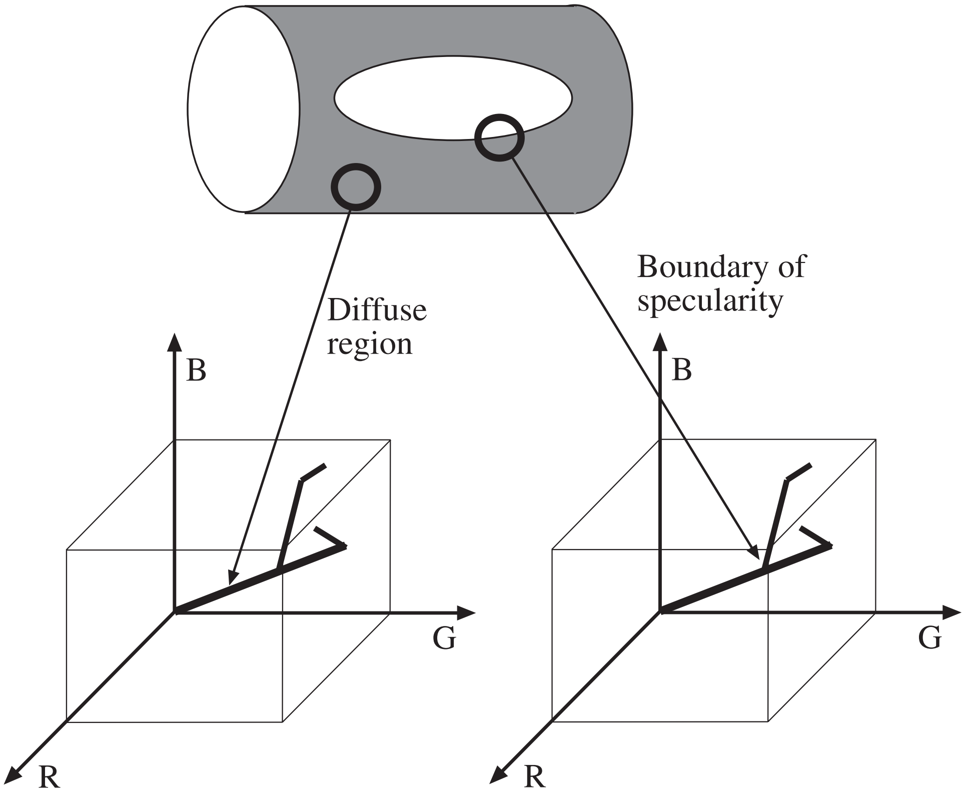


Figure 3.16. The linear clusters produced by specularities on plastic objects can be found by reasoning about windows of image pixels. In a world of plastic objects on a black background, a background window produces a region of pixels that are point-like in colour space — all pixels have the same colour. A window that lies along the body produces a line-like cluster of points in colour space, because the intensity varies but the colour does not. At the boundary of a specularity, windows produce plane-like clusters, because points are a weighted combination of two different colours (the specular and the body colour). Finally, at interior of a specular region, the windows can produce volume-like clusters, because the camera saturates, and the extent of the window can include both the boundary style window points and the saturated points. Whether a region is line-like, plane-like or volume like can be determined easily by looking at the eigenvalues of the covariance of the pixels.

3.5 Surface Colour from Image Colour

The colour of light arriving at a camera is determined by two factors: firstly, the spectral reflectance of the surface that the light is leaving, and secondly, the spectral radiance of the light falling on that surface. The colour of the light falling on surfaces can vary very widely — from blue fluorescent light indoors, to warm orange tungsten lights, to orange or even red light at sunset — so that the colour of the light arriving at the camera can be quite a poor representation of the colour of the surfaces being viewed (figures 3.17, 3.18, 3.19 and 3.20)

It would be attractive to have a **colour constancy** algorithm that could take an image, discount the effect of the light, and report the actual colour of the surfaces being viewed. Colour constancy is an interesting subproblem that has the flavour of a quite general vision problem: we are determining some parameters of the world from ambiguous image measurements; we need to use a model to disentangle these measurements; and we should like to be able to report more than one solution.

3.5.1 Surface Colour Perception in People

There is some form of colour constancy algorithm in the human vision system. People are often unaware of this, and inexperienced photographers are sometimes surprised that a scene photographed indoors under fluorescent lights has a blue cast, while the same scene photographed outdoors may have a warm orange cast.

It is common to distinguish between colour constancy — which is usually thought of in terms of intensity independent descriptions of colour like hue and saturation — and **lightness constancy**, the skill that allows humans to report whether a surface is white, grey or black (the **lightness** of the surface) despite changes in the intensity of illumination (the **brightness**). Colour constancy is neither perfectly accurate [], nor unavoidable. Humans can report:

- the colour a surface would have in white light (often called **surface colour**);
- colour of the light arriving at the eye, a skill that allows artists to paint surfaces illuminated by coloured lighting [];
- and sometimes, the colour of the light falling on the surface [].

All of these reports could be by-products of a colour constancy process.

The colorimetric theories of Section 3.3 can predict the colour an observer will perceive when shown an isolated spot of light of a given power spectral distribution. The human colour constancy algorithm appears to obtain cues from the structure of complex scenes, meaning that predictions from colorimetric theories can be wildly inaccurate if the spot of light is part of a larger, complex scene. Edwin Land's demonstrations [**?**] (which are illustrated in Figure 3.21) give convincing examples of this effect. It is surprisingly difficult to predict what colours a human will see in a complex scene [**?**; **?**]; this is one of the many difficulties that make it hard to produce really good colour reproduction systems (section **??**).

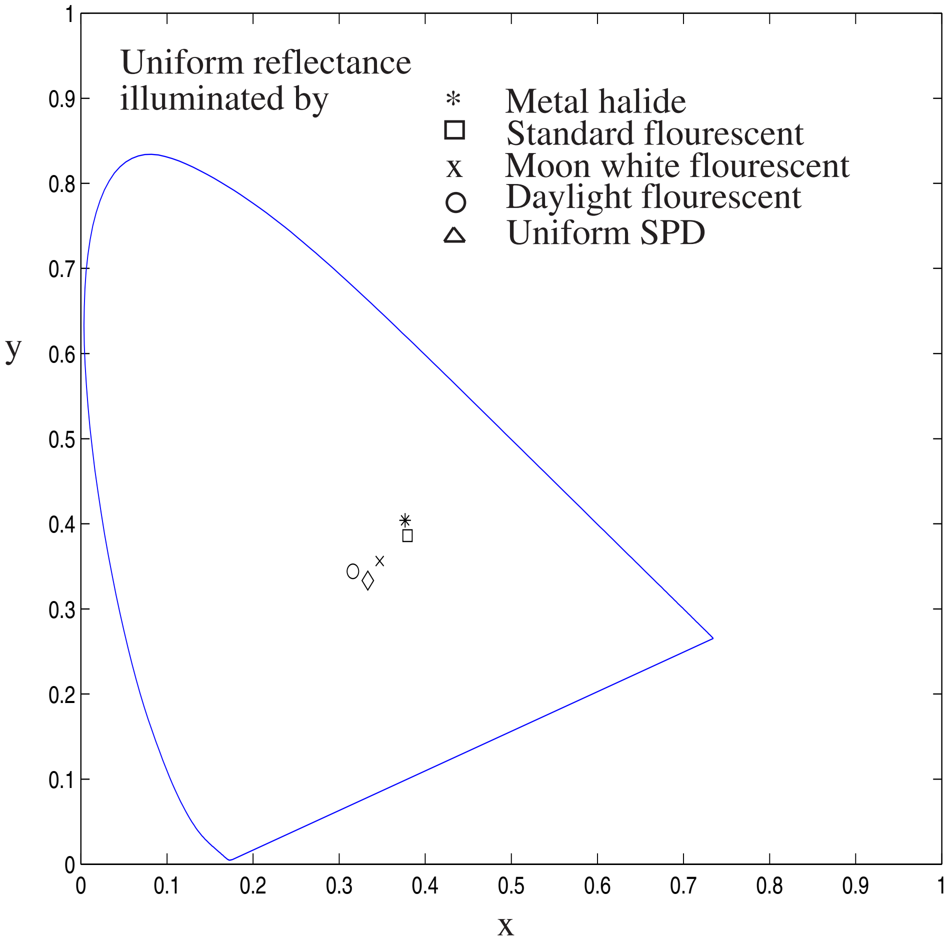


Figure 3.17. Light sources can have quite widely varying colours. This figure shows the colour of the four light sources of figure 3.4, compared with the colour of a uniform spectral power distribution, plotted in CIE x, y coordinates.

Human competence at colour constancy is surprisingly poorly understood. The main experiments on humans [**?**; **?**; **?**] do not explore all circumstances and it is not known, for example, how robust colour constancy is or the extent to which high-level cues contribute to colour judgements. Little is known about colour constancy in other animals — except that goldfish have it [**?**]. Colour constancy clearly fails — otherwise there would be no film industry — but the circumstances under which it fails are not well understood. There is a large body of data on surface lightness perception for achromatic stimuli. Since the brightness of a surface varies with its

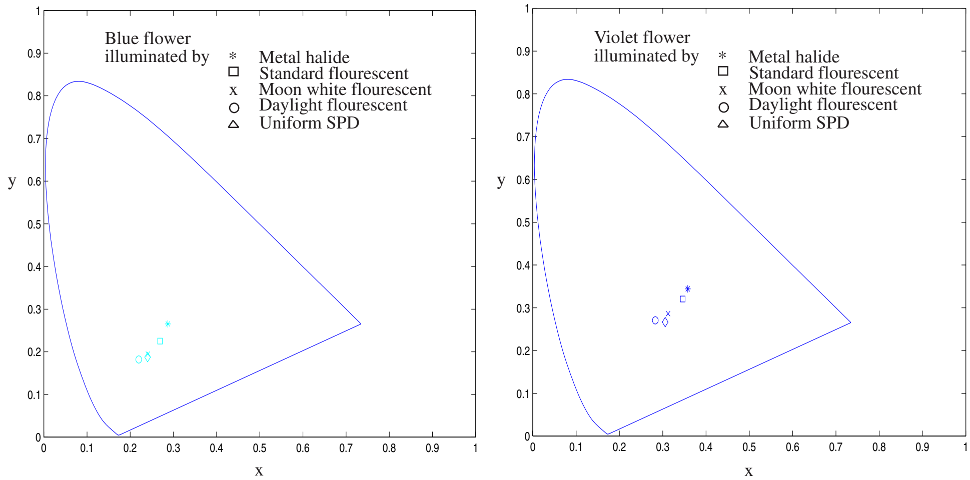


Figure 3.18. Surfaces have significantly different colours when viewed under different lights. These figures show the colours taken on by the blue flower and the violet flower of figure 3.1, when viewed under the four different sources of figure 3.4 and under a uniform spectral power distribution.

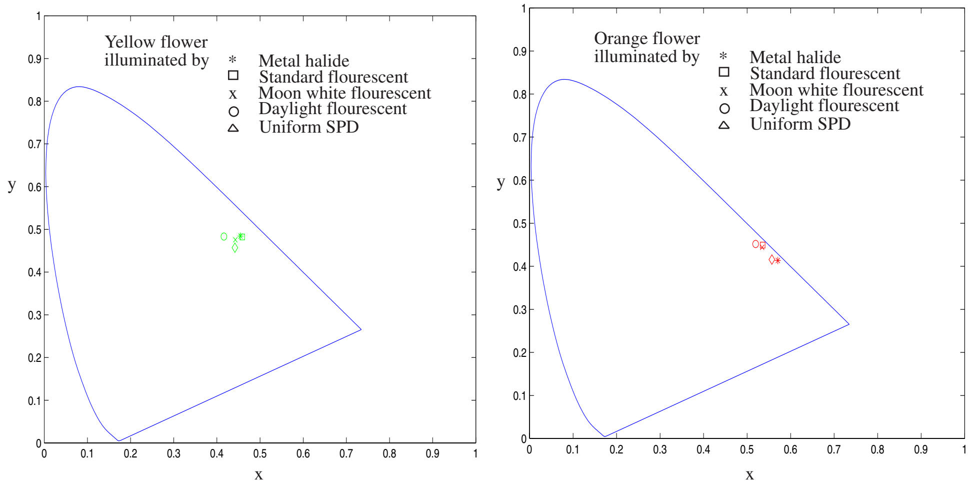


Figure 3.19. Surfaces have significantly different colours when viewed under different lights. These figures show the colours taken on by the yellow flower and the orange flower of figure 3.1, when viewed under the four different sources of figure 3.4 and under a uniform spectral power distribution.

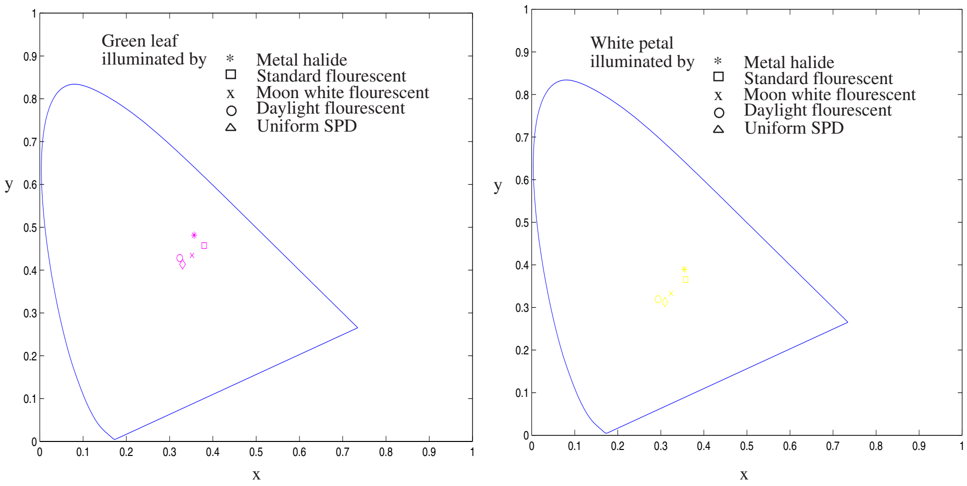


Figure 3.20. Surfaces have significantly different colours when viewed under different lights. These figures show the colours taken on by the white petal figure 3.1 and one of the leaves of figure 3.2, when viewed under the four different sources of figure 3.4 and under a uniform spectral power distribution.

orientation as well as with the intensity of the illuminant, one would expect that human lightness constancy would be poor: it is in fact extremely good over a wide range of illuminant variation [**?**].

3.5.2 Inferring Lightness

There is a lot of evidence that human lightness constancy involves two processes: one compares the brightness of various image patches, and uses this comparison to determine which patches are lighter and which darker; the second establishes some form of absolute standard to which these comparisons can be referred (e.g. [**?**]). We will describe lightness algorithms first, because they tend to be simpler than colour constancy algorithms.

A Simple Model of Image Brightness

The radiance arriving at a pixel depends on the illumination of the surface being viewed, its BRDF, its configuration with respect to the source and the camera responses. The situation is considerably simplified by assuming that the scene is plane and frontal; that surfaces are Lambertian; and that the camera responds linearly to radiance.

This yields a model of the camera response C at a point $\boldsymbol{X}$ as the product of an

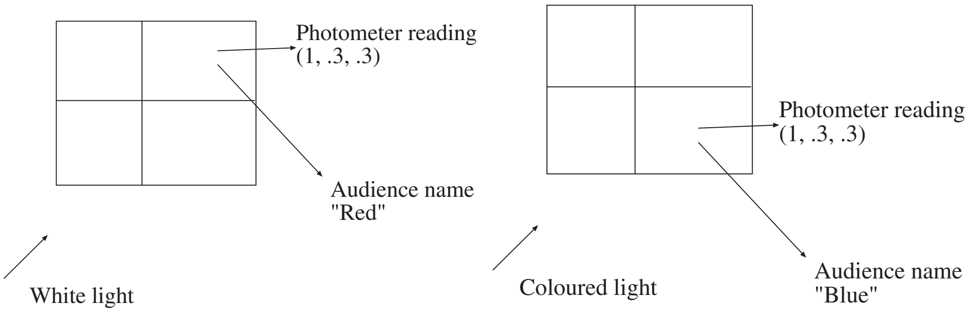


Figure 3.21. Land showed an audience a quilt of rectangles of flat coloured papers - since known as a Mondrian, for a purported resemblance to the work of that artist - illuminated using three slide projectors, casting red, green and blue light respectively. He used a photometer to measure the energy leaving a particular spot in three different channels, corresponding to the three classes of receptor in the eye. He recorded the measurement, and asked the audience to name the patch - say the answer was “red”. Land then adjusted the slide projectors so that some other patch reflected light that gave the same photometer measurements, and asked the audience to name that patch. The reply would describe the patch’s colour in white light - if the patch looked blue in white light, the answer would be “blue”. In later versions of this demonstration, Land put wedge-shaped neutral density filters into the slide-projectors, so that the colour of the light illuminating the quilt of papers would vary slowly across the quilt. Again, although the photometer readings vary significantly from one end of a patch to another, the audience sees the patch as having a constant colour.

illumination term, an albedo term and a constant that comes from the camera gain

$$C(\boldsymbol{x}) = k_c I(\boldsymbol{x}) \rho(\boldsymbol{x})$$

If we take logarithms, we get

$$\log C(\boldsymbol{x}) = \log k_c + \log I(\boldsymbol{x}) + \log \rho(\boldsymbol{x})$$

A second set of assumptions comes into play here.

- Firstly, we assume that albedoes change only quickly over space — this means that a typical set of albedoes will look like a collage of papers of different greys. This assumption is quite easily justified: firstly, there are relatively few continuous changes of albedo in the world (the best example occurs in ripening fruit); and secondly, changes of albedo often occur when one object occludes another (so we would expect the change to be fast). This means that spatial derivatives of the term $\log \rho(\boldsymbol{x})$ are either zero (where the albedo is constant) or large (at a change of albedo).
- Secondly, illumination changes only slowly over space. This assumption is somewhat realistic: for example, the illumination due to a point source will

> change relatively slowly unless the source is very close — so the sun is a source that is particularly good for this example; as another example, illumination inside rooms tends to change very slowly, because the white walls of the room act as area sources. This assumption fails dramatically at shadow boundaries however; we will have to see these as a special case, and assume that either there are no shadow boundaries, or that we know where they are.

Recovering Lightness from the Model

It is relatively easy to build algorithms that use our model. The earliest algorithm, Land's Retinex algorithm [**?**], has fallen into disuse. A natural approach is to differentiate the log transform, throw away small gradients, and then "integrate" the results [**?**]. There is a constant of integration missing, so lightness ratios are available, but absolute lightness measurements are not. Figure 3.22 illustrates the process for a one-dimensional example, where differentiation and integration are easy.

This approach can be extended to two dimensions as well. Differentiating and thresholding is easy: at each point, we estimate the magnitude of the gradient, and if the magnitude is less than some threshold, we set the gradient vector to zero, else we leave it alone. The difficulty is in integrating these gradients to get the log albedo map. The thresholded gradients may not be the gradients of an image, because the mixed second partials may not be equal (integrability again; compare with section 2.5.2).

The problem can be rephrased as a minimization problem: choose the log albedo map whose gradient is most like the thresholded gradient. This is a relatively simple problem, because computing the gradient of an image is a linear operation. The x-component of the thresholded gradient is scanned into a vector $\boldsymbol{p}$ and the y-component is scanned into a vector $\boldsymbol{q}$. We write the vector representing log-albedo as $\boldsymbol{l}$. Now the process of forming the x derivative is linear, and so there is some matrix $\mathcal{M}_x$ such that $\mathcal{M}_x\boldsymbol{l}$ is the x derivative; for the y derivative, we write the corresponding matrix $\mathcal{M}_y$.

The problem becomes to find the vector $\boldsymbol{l}$ that minimizes

$$| \mathcal{M}_x\boldsymbol{l} - \boldsymbol{p} |^2 + | \mathcal{M}_y\boldsymbol{l} - \boldsymbol{q} |^2$$

This is a quadratic minimisation problem, and the answer can be found by a linear process. Some special tricks are required, because adding a constant vector to $\boldsymbol{l}$ cannot change the derivatives, so the problem does not have a unique solution. We explore the minimisation problem in the exercises.

The constant of integration needs to be obtained from some other assumption. There are two obvious possibilities:

- we can assume that the *brightest patch is white*;
- we can assume that the *average lightness is constant.*

We explore the consequences of these models in the exercises.

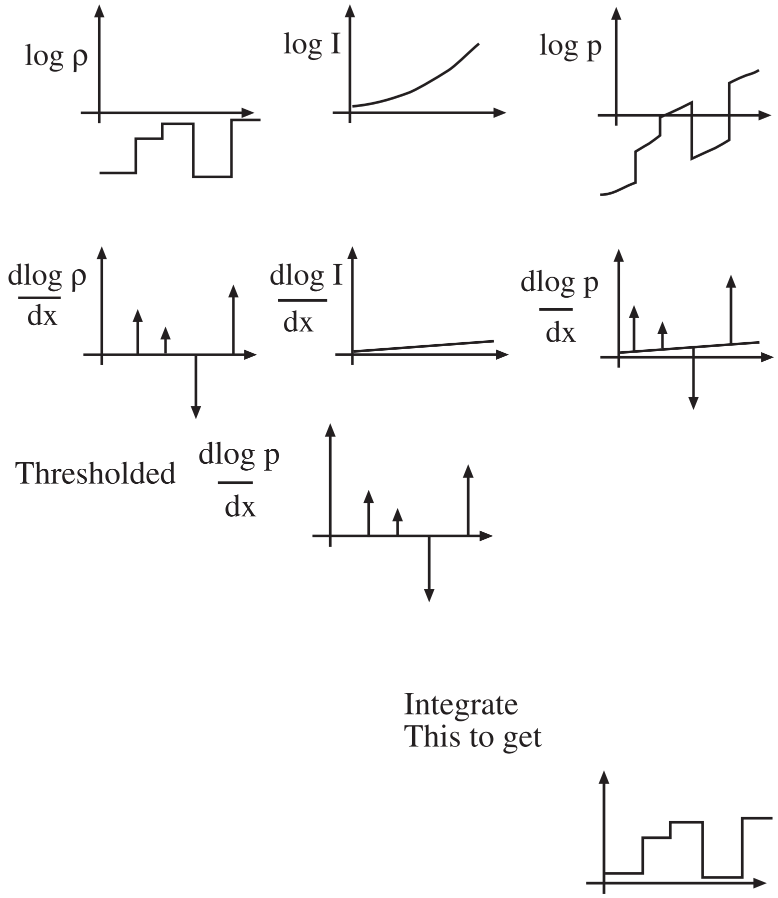


Figure 3.22. The lightness algorithm is easiest to illustrate for a 1D image. In the top row, the graph on the left shows $\log \rho(x)$; that on the center $\log I(x)$ and that on the right their sum which is $\log C$. The log of image intensity has large derivatives at changes in surface reflectance and small derivatives when the only change is due to illumination gradients. Lightness is recovered by differentiating the log intensity, thresholding to dispose of small derivatives, and then integrating, at the cost of a missing constant of integration.

3.5.3 A Model for Image Colour

To build a colour constancy algorithm, we need a model to interpret the colour of pixels. By suppressing details in the physical models of Chapters **??** and above, we can model the value at a pixel as:

$$\boldsymbol{C}(\boldsymbol{x}) = g_d(\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x}) + g_s(\boldsymbol{x})\boldsymbol{s}(\boldsymbol{x}) + \boldsymbol{i}(\boldsymbol{x})$$

```
Form the gradient of the log of the image

At each pixel, if the gradient magnitude is below
  a threshold, replace that gradient with zero

Reconstruct the log-albedo by solving the minimization
problem described in the text

Obtain a constant of integration

Add the constant to the log-albedo, and exponentiate
```

Algorithm 3.1: *Determining the Lightness of Image Patches*

In this model

- $\boldsymbol{d}(\boldsymbol{x})$ is the *image* colour of an equivalent *flat* frontal surface viewed under the same light;
- $g_d(\boldsymbol{x})$ is a term that varies over space and accounts for the change in brightness due to the orientation of the surface;
- $\boldsymbol{s}(\boldsymbol{x})$ is the *image* colour of the specular reflection from an equivalent *flat* frontal surface;
- $g_s(\boldsymbol{x})$ is a term that varies over space and accounts for the change in the amount of energy specularly reflected;
- and $\boldsymbol{i}(\boldsymbol{x})$ is a term that accounts for coloured interreflections, spatial changes in illumination, and the like.

We are primarily interested in information that can be extracted from colour at a local level, and so we are ignoring the detailed structure of the terms $g_d(\boldsymbol{x})$ and $\boldsymbol{i}(\boldsymbol{x})$. Nothing is known about how to extract information from $\boldsymbol{i}(\boldsymbol{x})$; all evidence suggests that this is very difficult. The term can sometimes be quite small with respect to other terms and usually changes quite slowly over space. We shall ignore this term, and so must assume that it is small (or that its presence does not disrupt our algorithms too severely).

Specularities are small and bright, and can be found using these properties (section 3.4). In principle, we could use the methods of that section to generate new images without specularities. This brings us to the term $g_d(\boldsymbol{x})\boldsymbol{d}(\boldsymbol{x})$ in the model above. Assume that $g_d(\boldsymbol{x})$ is a constant, so we are viewing a flat, frontal surface.

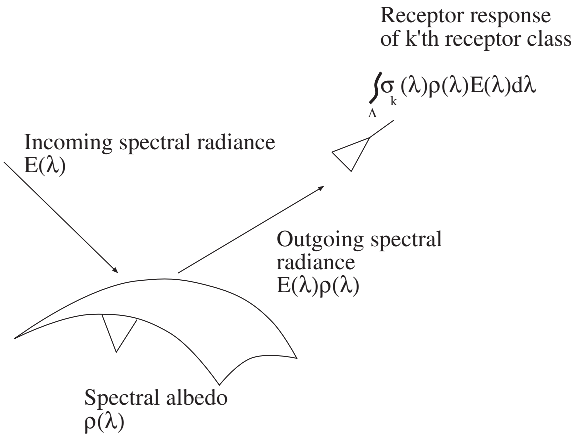


Figure 3.23. If a patch of perfectly diffuse surface with diffuse spectral reflectance $\rho(\lambda)$ is illuminated by a light whose spectrum is $E(\lambda)$, the spectrum of the reflected light will be $\rho(\lambda)E(\lambda)$ (multiplied by some constant to do with surface orientation, which we have already decided to ignore). Thus, if a photoreceptor of the k'th type sees this surface patch, its response will be: $p_k = \int_\Lambda \sigma_k(\lambda)\rho(\lambda)E(\lambda)d\lambda$ where Λ is the range of all relevant wavelengths and $\sigma_k(\lambda)$ is the sensitivity of the k'th photoreceptor.

The resulting term, $\boldsymbol{d}(\boldsymbol{x})$, models the world as a collage of flat, frontal diffuse coloured surfaces. We shall assume that there is a single illuminant that has a constant colour over the whole image. This term is a conglomeration of illuminant, receptor and reflectance information. It is impossible to disentangle completely in a realistic world. However, current algorithms can make quite usable estimates of surface colour from image colours, given a well populated world of coloured surfaces and a reasonable illuminant.

Finite-Dimensional Linear Models

The term $\boldsymbol{d}(\boldsymbol{x})$ is results from interactions between the spectral irradiance of the source, the spectral albedo of the surfaces, and the camera sensitivity. We need a model to account for these interactions. If a patch of perfectly diffuse surface with diffuse spectral reflectance $\rho(\lambda)$ is illuminated by a light whose spectrum is $E(\lambda)$, the spectrum of the reflected light will be $\rho(\lambda)E(\lambda)$ (multiplied by some constant to do with surface orientation, which we have already decided to ignore).

Thus, if a photoreceptor of the k'th type sees this surface patch, its response will be:

$$p_k = \int_\Lambda \sigma_k(\lambda)\rho(\lambda)E(\lambda)d\lambda$$

where Λ is the range of all relevant wavelengths and $\sigma_k(\lambda)$ is the sensitivity of the k'th photoreceptor (figure 3.23).

This response is linear in the surface reflectance and linear in the illumination, which suggests using linear models for the families of possible surface reflectances and illuminants. A **finite-dimensional linear model** models surface spectral albedoes and illuminant spectral irradiance as a weighted sum of a finite number of basis functions. We need not use the same bases for reflectances and for illuminants.

If a finite-dimensional linear model of surface reflectance is a reasonable description of the world, any surface reflectance can be written as

$$\rho(\lambda) = \sum_{j=1}^{n} r_j \phi_j(\lambda)$$

where the $\phi_j(\lambda)$ are the basis functions for the model of reflectance, and the r_j vary from surface to surface.

Similarly, if a finite-dimensional linear model of the illuminant is a reasonable model, any illuminant can be written as

$$E(\lambda) = \sum_{i=1}^{m} e_i \psi_i(\lambda)$$

where the $\psi_i(\lambda)$ are the basis functions for the model of illumination.

When both models apply, the response of a receptor of the k'th type is:

$$\begin{aligned} p_k &= \int \sigma_k(\lambda) \left(\sum_{j=1}^{n} r_j \phi_j(\lambda) \right) \left(\sum_{i=1}^{m} e_i \psi_i(\lambda) \right) d\lambda \\ &= \sum_{i=1, j=1}^{m,n} e_i r_j \left(\int \sigma_k(\lambda) \phi_j(\lambda) \psi_i(\lambda) \right) d\lambda \\ &= \sum_{i=1, j=1}^{m,n} e_i r_j g_{ijk} \end{aligned}$$

where we expect that the $g_{ijk} = \int \sigma_k(\lambda)\phi_j(\lambda)\psi_i(\lambda) d\lambda$ are known, as they are components of the world model (they can be learned from observations; see the exercises).

3.5.4 Surface Colour from Finite Dimensional Linear Models

Each of the indexed terms can be interpreted as components of a vector, and we shall use the notation $\boldsymbol{p}$ for the vector with k'th component p_k, etc. We could represent surface colour either directly by the vector of coefficients $\boldsymbol{r}$, or more indirectly by computing $\boldsymbol{r}$ and then determining what the surfaces would look like under white light. The latter representation is more useful in practice; among other things, the results are easy to interpret.

Normalizing Average Reflectance

Assume that the spatial average of reflectance in all scenes is constant and is known (for example, we might assume that all scenes have a spatial average of reflectance that is dull grey). In the finite-dimensional basis for reflectance we can write this average as

$$\sum_{j=1}^{n} \overline{r_j}\phi_j(\lambda)$$

Now if the average reflectance is constant, the average of the receptor responses must be constant too (the imaging process is linear), and the average of the response of the k'th receptor can be written as:

$$\overline{p_k} = \sum_{i=1,j=1}^{m,n} e_i g_{ijk} \overline{r_j}$$

If $\overline{\boldsymbol{p}}$ is the vector with k'th component $\overline{p}_k$ (using the notation above) and $\mathcal{A}$ is the matrix with k, i'th component

$$\sum_{j=1}^{n} \overline{r}_j g_{ijk}$$

then we can write the above expression as:

$$\overline{\boldsymbol{p}} = \mathcal{A}\boldsymbol{e}$$

For reasonable choices of receptors, the matrix $\mathcal{A}$ will have full rank, meaning that we can determine $\boldsymbol{e}$, which gives the illumination, *if* the finite dimensional linear model for illumination has the same dimension as the number of receptors. Of course, once the illumination is known, we can report the surface reflectance at each pixel, or correct the image to look as though it were taken under white light.

The underlying assumption that average reflectance is a known constant is dangerous, however, because it is usually not even close to being true. For example, if we assume that the average reflectance is a medium gray (a popular choice - see, for example, [**?**; **?**]), an image of a leafy forest glade will be reported as a collection of objects of various grays illuminated by green light. One way to try and avoid this problem is to change the average for different kinds of scenes [**?**] - but how do we decide what average to use? Another approach is to compute an average that is not a pure spatial average; one might, for example, average the colours that were represented by ten or more pixels, but without weighting them by the number of pixels present. It is hard to say in practice how well this approach could work; there is no experimental data in the literature.

Normalizing the Gamut

Not every possible pixel value can be obtained by taking images of real surfaces under white light. It is usually impossible to obtain values where one channel

```
Compute the average colour $\overline{\boldsymbol{p}}$ for the image

Compute $\boldsymbol{e}$ from $\overline{\boldsymbol{p}} = \mathcal{A}\boldsymbol{e}$

To obtain a version of the image under white light, $\boldsymbol{e}^w$:
  Now for each pixel, compute $\boldsymbol{r}$ from $p_k = \sum_{i=1,j=1} m, n e_i g_{ijk} r_j$

  Replace the pixel value with $p_k^w = \sum_{i=1,j=1} m, n e_i^w g_{ijk} r_j$
```

Algorithm 3.2: *Colour Constancy from Known Average Reflectance*

responds strongly and others do not - for example, 255 in the red channel and 0 in the green and blue channels. This means that the gamut of an image - the collection of all pixel values - contains information about the light source. For example, if one observes a pixel that has value $(255, 0, 0)$, then the light source is likely to be red in colour.

If an image gamut contains two pixel values, say $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, then it must be possible to take an image *under the same illuminant* that contains the value $t\boldsymbol{p}_1 + (1-t)\boldsymbol{p}_2$ for $0 \leq t \leq 1$ (because we could mix the colorants on the surfaces). This means that the convex hull of the image gamut contains the illuminant information. These constraints can be exploited to constrain the colour of the illuminant.

Write G for the convex hull of the gamut of the given image, W for the convex hull of the gamut of an image of many different surfaces under white light, and $\mathcal{M}_{\boldsymbol{e}}$ for the map that takes an image seen under illuminant $\boldsymbol{e}$ to an image seen under white light. Then the only illuminants we need to consider are those such that $\mathcal{M}_{\boldsymbol{e}}(G) \in W$. This is most helpful if the family $\mathcal{M}_{\boldsymbol{e}}$ has a reasonable structure; one natural example is to assume that changes in one illuminant parameter affect only the response of a single receptor. In turn, this means that elements of $\mathcal{M}_{\boldsymbol{e}}$ are diagonal matrices.

In the case of finite dimensional linear models, $\mathcal{M}_{\boldsymbol{e}}$ depends linearly on $\boldsymbol{e}$, so that the family of illuminants that satisfy the constraint is also convex. This family can be constructed by intersecting a set of convex hulls, each corresponding to the family of maps that takes a hull vertex of G to some point inside W (or we could write a long series of linear constraints on $\boldsymbol{e}$).

Once we have formed this family, it remains to find an appropriate illuminant. There are a variety of possible strategies: if something is known about the likelihood of encountering particular illuminants, then one might choose the most likely; assuming that most pictures contain many different coloured surfaces leads to the choice of illuminant that makes the restored gamut the largest (which is the approach that generated the results of figure **??**); or one might use other constraints

```
Obtain the gamut $W$ of many images of
  many different coloured surfaces under white
  light (this is the convex
  hull of all image pixel values)

Obtain the gamut $G$ of the image (this is the convex
  hull of all image pixel values)

Obtain every element of the family of illuminant maps $\mathcal{M}_{\boldsymbol{e}}$
such that $\mathcal{M}_{\boldsymbol{e}} G \in W$
this represents all possible illuminants

Choose some element of this family, and apply
it to every pixel in the image
```

Algorithm 3.3: *Colour Constancy by Gamut Mapping*

on illuminants - for example, all the illuminants must have non-negative energy at all wavelengths - to constrain the set even further [**?**].

3.6 Notes

The use of colour in computer vision is surprisingly primitive. One difficulty is some legitimate uncertainty about what it is good for. John Mollon's remark that the primate colour system could be seen as an innovation of some kinds of fruiting tree [] is one explanation, but it is not much help.

3.6.1 Trichromacy and Colour Spaces

Up until quite recently, there was no conclusive explanation of why trichromacy applied, although it was generally believed to be due to the presence of three different types of colour receptor in the eye. Work on the genetics of photoreceptors by Nathans *et al.* can be interpreted as confirming this hunch (see []), though a full explanation is still far from clear because this work can also be interpreted as suggesting many individuals have more than three types of photoreceptor [].

There is an astonishing number of colour spaces and colour appearance models available. We discourage the practice of publishing papers that compare colour *spaces* for, say, segmentation, because the spaces are within one-one transformations of one another. The important issue is not in what coordinate system one measures colour, but how one counts the difference — so colour metrics may still bear some thought.

Colour metrics are an old topic; usually, one fits a metric tensor to MacAdam ellipses. The difficulty with this approach is that a metric tensor carries the strong implication that you can measure differences over large ranges by integration, whereas it is very hard to see large range colour comparisons as meaningful. Another concern is that the weight observers place on a difference in a Maxwellian view and the semantic significance of a difference in image colours are two very different things.

3.6.2 Lightness and Colour Constancy

There has not been much recent study of lightness constancy algorithms. The basic idea is due to Land []; his work was formalised for the computer vision community by Horn []; and a variation on Horn's algorithm was constructed by Blake [**?**]. The techniques are not as widely used as they should be, particularly given that there is some evidence they produce useful information on real images [**?**]. Classifying illumination vs albedo simply by looking at the magnitude of the gradient is crude, and ignores at least one important cue (very large changes must be illumination, however fast they occur); there is significant room for improvement.

The most significant case in colour constancy occurs when there are three classes of photoreceptor; others have been studied [**?**; **?**; **?**; **?**; **?**], but this is mainly an excuse to do linear algebra.

Finite-dimensional linear models for spectral reflectances can be supported by an appeal to surface physics, as spectral absorbtion lines are thickened by solid state effects. The main experimental justifications for finite-dimensional linear models of surface reflectance are Cohen's [**?**] measurements of the surface reflectance of a selection of standard reference surfaces known as **Munsell chips**, and Krinov's [**?**] measurements of a selection of natural objects. Cohen [**?**] performed a principal axis decomposition of his data, to obtain a set of basis functions, and Maloney [**?**] fitted weighted sums of these functions to Krinov's date to get good fits with patterned deviations. The first three principal axes explained in each case a very high percentage of the sample variance (near 99 %), and hence a linear combination of these functions fitted all the sampled functions rather well. More recently, Maloney [**?**] fitted Cohen's basis vectors to a large set of data, including Krinov's data, and further data on the surface reflectances of Munsell chips, and concluded that the dimension of an accurate model of surface reflectance was of the order of five or six.

On surfaces like plastics, the specular component of the reflected light is the same colour as the illuminant. If we can identify specular regions from such objects in the image, the colour of the illuminant is known. This idea has been popular for a long time[2]. Recent versions of this idea appear in, for example, [**?**; **?**; **?**; **?**].

There is surprisingly little work on colour constancy that unifies a study of the spatial variation in illumination with solutions for surface colour, which is why we were reduced to ignoring a number of terms in our colour model. There is substantial room for research here, too.

---

[2] Judd [**?**] writing in 1960 about early German work in surface colour perception refers to it as "a more usual view".

3.6.3 Colour in Recognition

As future chapters will show, it is quite tricky to build systems that use object colour to help in recognition. A variety of effects cause image colours to be poor measurements of surface colour. Uniform colour spaces offer some help here, if we are willing to swallow a fairly loose evolutionary argument: it is worth understanding the colour differences that humans recognise, because they are adapted to measurements that are useful.

3.7 Assignments

Exercises

1. Sit down with a friend and a packet of coloured papers, and compare the colour names that you use. You will need a large packet of papers — one can very often get collections of coloured swatches for paint, or for the Pantone colour system very cheaply. The best names to try are basic colour names — the terms red, pink, orange, yellow, green, blue, purple, brown, white, gray and black, which (with a small number of other terms) have remarkable canonical properties that apply widely across different languages [**?**; **?**; **?**]. You will find it surprisingly easy to disagree on which colours should be called blue and which green, for example.

2. Derive the equations for transforming from RGB to CIE XYZ, and back. This is a linear transformation. It is sufficient to write out the expressions for the elements of the linear transformation — you don't have to look up the actual numerical values of the colour matching functions.

3. Linear colour spaces are obtained by choosing primaries and then constructing colourmatching functions for those primaries. Show that there is a linear transformation that takes the coordinates of a colour in one linear colour space to those in another; the easiest way to do this is to write out the transformation in terms of the colourmatching functions.

4. Exercise 3 means that, in setting up a linear colour space, it is possible to choose primaries arbitrarily, but there are constraints on the choice of colour matching functions. Why? What are these constraints?

5. Two surfaces that have the same colour under one light and different colours under another are often referred to as *metamers.* An *optimal colour* is a spectral reflectance or radiance that has value 0 at some wavelengths and 1 at others. Though optimal colours don't occur in practice, they are a useful device (due to Ostwald) for explaining various effects.

    - use optimal colours to explain how metamerism occurs.

- given a particular spectral albedo, show that there are an infinite number of metameric spectral albedoes.
- use optimal colours to construct an example of surfaces that look very different under one light (say, red and green) and the same under another.
- use optimal colours to construct an example of surfaces that swop apparent colour when the light is changed (i.e. surface one looks red and surface two looks green under light one, and surface one looks green and surface two looks red under light two).

6. You have to map the gamut for a printer to that of a monitor. There are colours in each gamut that do not appear in the other. Given a monitor colour that can't be reproduced exactly, you could choose the printer colour that is closest. Why is this a bad idea for reproducing images? Would it work for reproducing "business graphics" (bar charts, pie charts, and the like which all consist of many differernt large blocks of a single colour)?

7. *Volume colour* is a phenomenon associated with translucent materials that are coloured — the most attractive example is a glass of wine. The colouring comes from different absorption coefficients at different wavelengths. Explain (1) why a small glass of sufficiently deeply coloured red wine (a good Cahors, or Gigondas) looks black (2) why a big glass of lightly coloured red wine also looks black. Experimental work is optional.

8. (This exercise requires some knowledge of numerical analysis). In section 3.5.2, we set up the problem of recovering the log-albedo for a set of surfaces as one of minimizing

   $$| \mathcal{M}_x \boldsymbol{l} - \boldsymbol{p} |^2 + | \mathcal{M}_y \boldsymbol{l} - \boldsymbol{q} |^2$$

   where $\mathcal{M}_x$ forms the x derivative of $\boldsymbol{l}$ and $\mathcal{M}_y$ forms the y derivative (i.e. $\mathcal{M}_x \boldsymbol{l}$ is the x-derivative).

   - We asserted that $\mathcal{M}_x$ and $\mathcal{M}_y$ existed. Use the expression for forward differences (or central differences, or any other difference approximation to the derivative) to form these matrices. Almost every element is zero.
   - The minimisation problem can be written in the form

     $$\text{choose } \boldsymbol{l} \text{ to minimize } (\mathcal{A}\boldsymbol{l} + \boldsymbol{b})^T (\mathcal{A}\boldsymbol{l} + b)$$

     Determine the values of $\mathcal{A}$ and $\boldsymbol{b}$, and show how to solve this general problem. You will need to keep in mind that $\mathcal{A}$ does not have full rank, so you can't go inverting it.

9. In section 3.5.2, we mentioned two assumptions that would yield a constant of integration.

   - Show how to use these assumptions to recover an albedo map.

- For each assumption, describe a situation where it fails, and describe the nature of the failure. Your examples should work for cases where there are many different albedoes in view.

10. Read the book " Colour: Art and Science", by Lamb and Bourriau, Cambridge University Press, 1995.

Programming Assignments

1. Spectra for illuminants and for surfaces are available on the web (for example `http:whereisit?`). Fit a finite-dimensional linear model to a set of illuminants and surface reflectances using principal components analysis, render the resulting models, and compare your rendering with an exact rendering. Where do you get the most significant errors? why?

2. Print a coloured image on a colour inkjet printer using different papers and compare the result. It is particularly informative to (a) ensure that the driver knows what paper the printer will be printing on, and compare the variations in colours (which are ideally imperceptible) and (b) deceive the driver about what paper it is printing on (i.e. print on plain paper and tell the driver it is printing on photographic paper). Can you explain the variations you see? Why is photographic paper glossy?

3. Fitting a finite-dimensional linear model to illuminants and reflectances separately is somewhat ill-advised, because there is no guarantee that the *interactions* will be represented well (they're not accounted for in the fitting error). It turns out that one can obtain g_{ijk} by a fitting process that sidesteps the use of basis functions. Implement this procedure (which is described in detail in [**?**]), and compare the results with those obtained from the previous assignment.

4. Build a colour constancy algorithm that uses the assumption that the spatial average of reflectance is constant. Use finite-dimensional linear models. You can get values of g_{ijk} from your solution to exercise 3.

5. We ignore colour interreflections in our surface colour model. Do an experiment to get some idea of the size of colour shifts possible from colour interreflections (which are astonishingly big). Humans very seldom interpret colour interreflections as surface colour — speculate as to why this might be the case, using the discussion of the lightness algorithm as a guide.

6. Build a specularity finder along the lines described in section 3.4.

Part II

IMAGE MODELS

Chapter 4

GEOMETRIC IMAGE FEATURES

What is the image of a given geometric figure, for example a line, a plane, a sphere, or a general smooth surface? Answering this question is of course important if we are to interpret pictures of our three-dimensional world. As shown analytically in the Chapter 5, answering it is also easy for linear features such as points and lines: indeed, under perspective projection, points map onto points, and lines project (in general) onto lines. Geometrically, this is also obvious for points, and the image of a line is simply the intersection of the retina with the plane that contains this line and the pinhole (Figure 4.1(a)). In truth, things degenerate a bit for *exceptional views*: when a line passes through the pinhole, its image is reduced to a point (Figure 4.1(b)). Likewise, the image of a surface patch covers in general a finite area of the image plane, but the projection of a plane passing through the pinhole is reduced to a line. Still, almost any small perturbation of the viewpoint restores the usual projection pattern.

By the same argument, general viewpoint assumptions allow us to infer three-dimensional scene information from a single image: in particular, Figure 4.1(b) shows that two points that appear to coincide in the image usually also coincide in space since the set of pinhole positions aligned with two distinct points has a zero volume in the three-dimensional set of possible viewpoints.

More generally, the edges and vertices of a polyhedral solid project onto line segments and points of the image plane. A subset of these segments forms a polygon that bounds the image of the solid, corresponding to surface edges adjacent to faces whose normals point into opposite directions relative to the observer (Figure 4.2(a)). To assert stronger image properties requires additional hypotheses: for example, assuming a general viewpoint and a *trihedral world*, where all polyhedral vertices are formed by the intersection of exactly three planar faces, it is possible to show that the image edges joining at a vertex may only be arranged in four qualitatively distinct configurations, the so-called *arrow*, *fork*, *L-* and *T-junctions* (Figure 4.2(b)).

Using these junctions to construct three-dimensional interpretations of line-drawings is a time-honored artificial intelligence exercise that had its hayday in

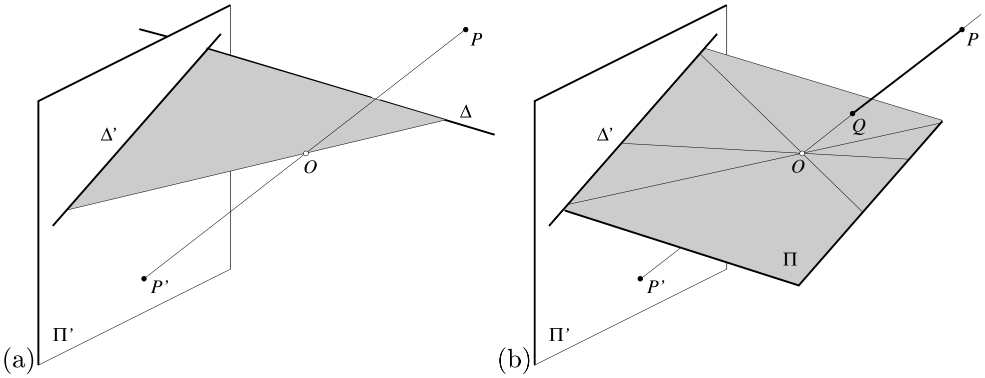


Figure 4.1. Point and line projections: (a) general case: the grey triangle represents the plane defined by the line Δ and the pinhole O; (b) degenerate case: the line supporting the segment PQ and passing through O projects onto the point P', and the plane Π passing through O projects onto the line Δ'.

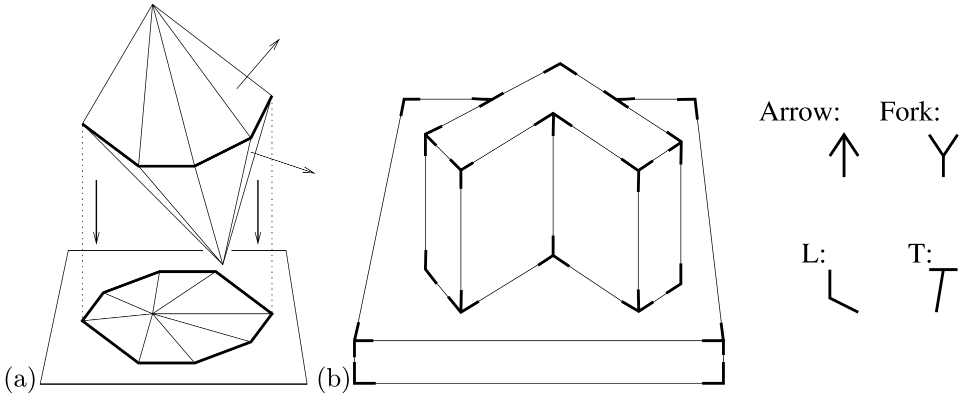


Figure 4.2. The images of polyhedral solids: (a) polygonal edges of the image of a polyhedron; (b) the image of a trihedral object and the corresponding junctions. Note that the solid shown in (a) is not trihedral, and the image junction corresponding to its top vertex is neither an arrow nor a fork, an L- or a T-junction.

the early seventies but has fallen out of favor because of the great difficulty in constructing a program that reliably extracts flawless line-drawings from images. This merely suggests that a purely bottom-up approach to image understanding may fail, but should not detract from the fundamental importance of understanding how lines, points and other simple geometric objects project into images.

What about more complex figures? It is easy to show that any (planar or twisted) curve that can be defined by polynomial equations (e.g., a conic section, which is defined by a quadratic equation) also projects onto a curve defined by polynomial equations. The case of curved surfaces is more interesting: consider for example a solid bounded by a smooth surface with a smooth reflectance function, observed by a perspective camera under a smooth illumination pattern. Ignoring shadows (e.g., considering a single point light source and a camera co-located far from the scene), the image is also smooth, except perhaps along *occlusion boundaries*, where the object surface turns away from the camera, and two points that project on opposite sides of these boundaries belong to spatially separate parts of the surface.

Intuitively, it is clear that occlusion boundaries form a set of image curves, called the *outline*, *silhouette*, or *contour* of the object (Figure 4.3(a)). As in the case of spheres discussed in Chapter **??**, these curves are formed by intersecting the retina with viewing cones (or cylinders in the case of orthographic projection) whose apex coincides with the pinhole and whose surface grazes the object along a second set of curves, called the *occluding contour*, or *rim*.

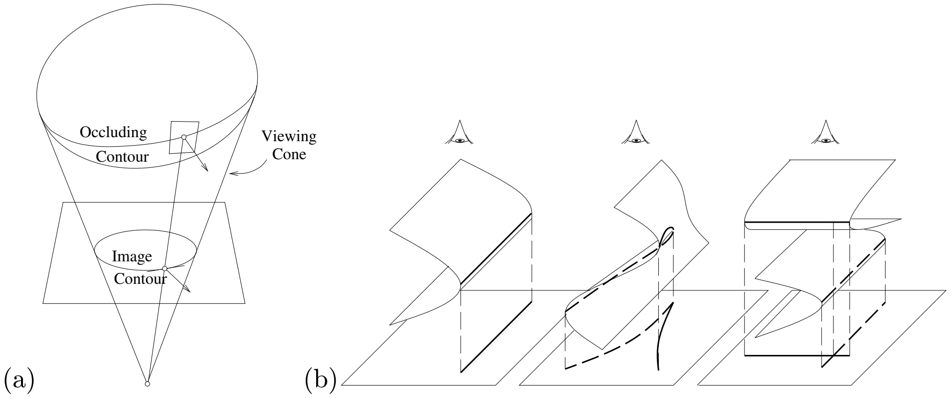


Figure 4.3. The images of solids bounded by smooth surfaces: (a) the occluding boundaries of a smooth surface; (b) contour components: folds, cusps, and T-junctions. Reprinted from [**?**], Figure 3.

This intuition can be made mathematically precise. In particular, it can be shown that the occluding contour is in general a smooth curve, formed by *fold* points, where the viewing ray is tangent to the surface, and a discrete set of *cusp* points where the ray is tangent to the occluding contour as well. The image contour is piecewise smooth, and its only singularities are a discrete set of *cusps*, formed by the projection of cusp points, and *T-junctions*, formed by the transversal superposition of pairs of fold points (Figure 4.3(b)). The intuitive meaning of these exotic terms should be pretty clear: a fold is a point where the surface folds away from its viewer, and a contour cusps at a point where it suddenly decides to turn back, following a different path along the same tangent (this is for transparent objects only: contours of opaque objects terminate at cusps, see Figure 4.3(b)). Likewise, two smooth pieces of contour cross at a T-junction (unless the object is opaque and one of the branches terminates at the junction).

Interestingly, attached shadows are delineated by the occluding contours associated with the light sources, and cast shadows are bounded by the corresponding object outlines (Figure 4.4). Thus we also know what they look like. (Caveat: the objects onto which shadows are cast may themselves have curved surfaces, which complicates things a great deal. However, the boundaries of attached shadows are really just occluding contours. Of course, light sources are rarely punctual, and this adds further difficulties..)

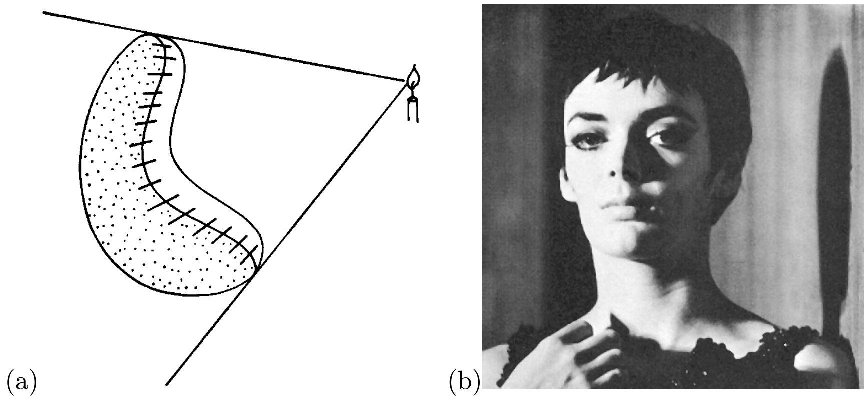


Figure 4.4. (a) Shadow boundaries and occluding contours. Reprinted from [?], Figure 157. (b) The actress Barbara Steele and the cast and attached shadows of her face. Reprinted from [?], p. 33.

It should not come as a surprise that the notion of occluding boundaries carries over to the polyhedral world: the image contour of a polyhedron is exactly the polygon bounding its image, and its occluding contour is formed by the edges separating faces that point in opposite directions relative to the observer (these are drawn as thicker line segments in Figure 4.2(a)). The image contour of a solid shape constrains it to lie within the associated viewing cone but does not reveal the depth of its occluding contour. In the case of solids bounded by smooth surfaces, the contour provides additional information: in particular, the plane defined by the eye and the tangent to the image contour is itself tangent to the surface. Thus the contour orientation determines the surface orientation along the occluding contour.

More generally, the rest of this chapter focuses on the geometric relationship between curved surfaces and their projections and on the type of information about surface geometry that can be infered from contour geometry (we will come back later to the case of polyhedral solids): for example, we will show that the contour curvature also reveals information about the surface curvature. In the mean time, let us start by introducing elementary notions of differential geometry that will provide a natural mathematical setting for our study. Differential geometry will prove useful again later in this book, when we study the changes in object appearance that stem

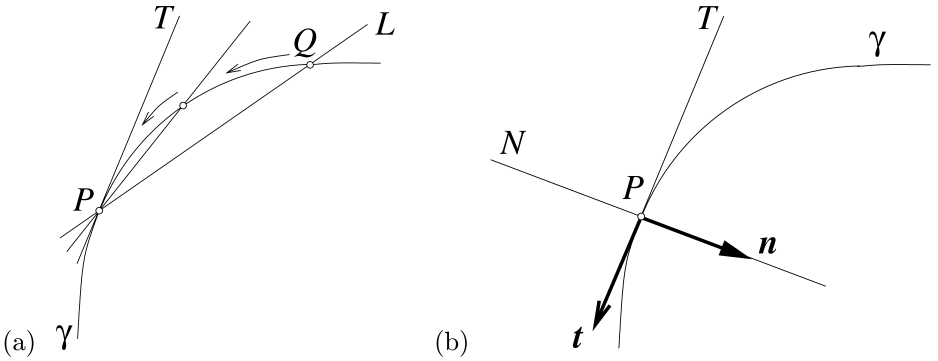


Figure 4.5. Tangents and normals: (a) definition of the tangent as the limit of secants; (b) the coordinate system defined by the (oriented) tangent and normal.

from viewpoint changes.

4.1 Elements of Differential Geometry

In this section we will present the rudiments of differential geometry necessary to understand the local relationship between light rays and solid objects. The topic of our discussion is of course technical, but we will attempt to stay at a fairly informal level, emphasizing descriptive over analytical geometry. We will also illustrate some of the concepts introduced with a simple study of surface specularities and what they reveal about surface shape. Following [?] we will limit our study of surfaces to those bounding compact solids in Euclidean space.

4.1.1 Curves

We start with the study of curves that lie in a plane. We will examine a curve γ in the immediate vicinity of some point P, and will assume that γ does not intersect itself, or, for that matter, terminate in P. If we draw a straight line L through P, it will (in general) intersect γ in some other point Q, defining a *secant* of this curve (Figure 4.5(a)). As Q moves closer to P, the secant L will rotate about P and approach a limit position T, called the *tangent line* to γ in P.

By construction, the tangent T has more intimate contact with γ than any other line passing through P. Let us now draw a second line N through P and perpendicular to L, and call it the *normal* to γ in P. Given an (arbitrary) choice for a unit *tangent vector* $\boldsymbol{t}$ along L, we can construct a right-handed coordinate frame whose origin is P and whose axes are $\boldsymbol{t}$ and a unit *normal vector* $\boldsymbol{n}$ along N.

This coordinate system is particularly well adapted to the study of the curve in the neighborhood of P: its axes divide the plane into four quadrants that can be numbered in counterclockwise order as shown in Figure 4.6, the first quadrant being chosen so it contains a particle traveling along the curve toward (and close to) the origin. In which quadrant will this particle end up just after passing P? As shown by Figure 4.6(a)-(d), there are four possible answers to this question, and they characterize the shape of the curve near P: we say that P is *regular* when the moving point ends up in the second quadrant, and *singular* otherwise. When the particle traverses the tangent and ends up in the third quadrant, P is called an *inflection* of the curve, and we say that P is a *cusp* of the first or second kind in the two remaining cases respectively. This classification is in fact independent of the orientation chosen for γ, and it turns out that almost all points of a general curve are regular, while singularities only occur at isolated points.

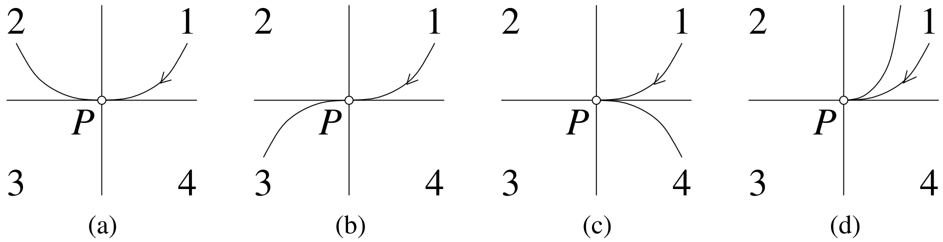


Figure 4.6. A classification of curve points: (a) a regular point; (b) an inflection; (c) a cusp of the first kind; (d) a cusp of the second kind. Note that the curve stays on the same side of the tangent at regular points.

As noted before, the tangent to a curve γ in P is the closest linear approximation of γ passing through this point. In turn, constructing the closest *circular* approximation will now allow us to define the *curvature* in P, another fundamental characteristic of the curve shape: consider a point P' as it approaches P along the curve, and let M denote the intersection of the normal lines N and N' in P and P' (Figure 4.7). As P' moves closer to P, M approaches a limit position C along the normal N, called the *center of curvature* of γ in P.

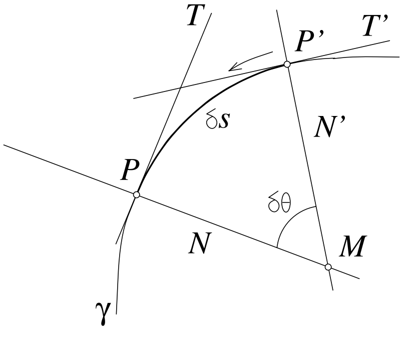


Figure 4.7. Definition of the center of curvature as the limit of the intersection of normal lines through neighbors of P.

At the same time, if $\delta\theta$ denotes the (small) angle between the normals N and N', and δs denotes the length of the (short) curve arc joining P and P', the ratio $\delta\theta/\delta s$ also approaches a definite limit κ, called the *curvature* of the curve in P, as δs nears zero. It turns out that κ is just the inverse of the distance r between C and P (this follows easily from the fact that $\sin u \approx u$ for small angles, see exercises). The circle centered in C with radius r is called the *circle of curvature* in P, and r is the *radius of curvature.*

It can also be shown that a circle drawn through P and two close-by points P' and P'' approaches the circle of curvature as P' and P'' move closer to P. This circle is indeed the closest circular approximation to γ passing through P. The curvature is zero at inflections, and the circle of curvature degenerates to a straight line (the tangent) there: inflections are the "flattest" points along a curve.

We will now introduce a device that will prove to be extremely important in the study of both curves and surfaces, the *Gauss map.* Let us pick an orientation for the curve γ and associate with every point P on γ the point Q on the unit circle S^1 where the tip of the associated normal vector meets the circle (Figure 4.8). The corresponding mapping from γ to S^1 is the Gauss map associated with γ.[1]

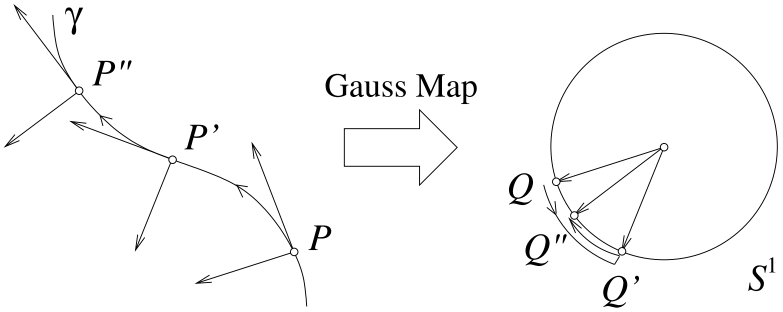


Figure 4.8. The Gaussian image of a plane curve. Observe how the direction of traversal of the Gaussian image reverses at the inflection P' of the curve. Also note that there are close-by points with parallel tangents/normals on either side of P'. The Gaussian image folds at the corresponding point Q'.

Let us have another look at the limiting process used to define the curvature. As P' approaches P on the curve, so does the Gaussian image Q' of P' approach the image Q of P. The (small) angle between N and N' is equal to the length of the arc joining Q and Q' on the unit circle. The curvature is therefore the limit of the ratio between the lengths of corresponding arcs of the Gaussian image and of the curve as both approach zero.

The Gauss map also provides an interpretation of the classification of curve points introduced earlier: consider a particle traveling along a curve and the corresponding motion of its Gaussian image. The direction of traversal of γ stays the same at regular points and inflections, but reverses for both types of cusps (Figure 4.6). On the other hand, the direction of traversal of the Gaussian image stays the same at regular points and cusps of the first kind, but it reverses at inflections and cusps of the second kind (Figure 4.8). This indicates a double covering of the unit circle near these singularities: we say that the Gaussian image *folds* at these points.

---

[1] The Gauss map could have been defined just as well by associating with each curve point the tip of its unit tangent on S^1. The two representations are equivalent in the case of planar curves. The situation will be different when we generalize the Gauss map to twisted curves and surfaces.

A conventional sign can be chosen for the curvature at every point of a plane curve γ by picking some orientation for this curve, and deciding, say, that the curvature will be positive when the center of curvature lies on the same side of γ as the tip of the oriented normal vector, and negative when these two points lie on opposite sides of γ. Thus the curvature changes sign at inflections, and reversing the orientation of a curve also reverses the sign of its curvature.

Twisted space curves are more complicated animals that their planar counterparts. Although the tangent can be defined as before as a limit of secants, there is now an infinity of lines perpendicular to the tangent at a point P, forming a *normal plane* to the curve at this point (Figure 4.9).

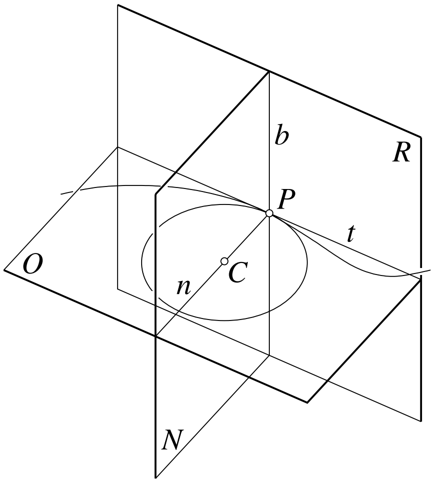


Figure 4.9. The local geometry of a space curve: N, O and R are respectively the normal, osculating, and rectifying plane; t, n and b are respectively the tangent, (principal) normal and binormal lines, and C is the center of curvature.

In general, a twisted curve does not lie in a plane in the vicinity of one of its points, but there exists a unique plane that lies closest to it. This is the *osculating plane*, defined as the limit of the plane containing the tangent line in P and some close-by curve point Q as the latter approaches P. We finish the construction of a local coordinate frame in P by drawing a *rectifying plane* through P, perpendicular to both the normal and osculating planes. The axes of this coordinate system, called *moving trihedron*, or *Frénet frame*, are the tangent, the *principal normal* formed by the intersection of the normal and osculating planes, and the *binormal* defined by the intersection of the normal and rectifying planes (Figure 4.9).

As in the planar case, the curvature of a twisted curve can be defined in a number of ways: as the inverse of the radius of the limit circle defined by three curve points as they approach each other (this circle of curvature lies in the osculating plane), as the limit ratio of the angle between the tangents at two close-by points and the distance separating these points as it approaches zero, etc. Likewise, the concept of Gaussian image can be extended to space curves, but this time the tips of the tangents, principal normals and binormals draw curves on a unit *sphere*. Note that it is not possible to give a meaningful sign to the curvature of a twisted curve: in general, such a curve does not have inflections, and its curvature is positive everywhere.

The curvature can be thought of as a measure of the rate of change of the tangent direction along a curve. It is also possible to define the rate of change of the osculating plane direction along a twisted curve: consider two close-by points P and P' on the curve; we can measure the angle between their osculating planes, or equivalently between the associated binormals, and divide this angle by the distance between the two points. The limit of this ratio as P' approaches P is called the *torsion* of the curve in P. Not surprisingly, its inverse is the limit of the ratio between the lengths of corresponding arcs on the curve and the spherical image of the binormals.

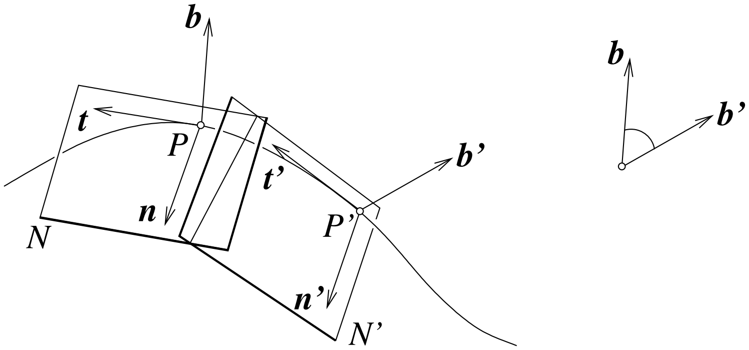


Figure 4.10. Geometric definition of the torsion as the limit, as both quantities approach zero, of the ratio obtained by dividing the angle between the binormals by the distance between the associated surface points.

A space curve can be oriented by considering it as the trajectory of a moving particle and picking a direction of travel for this particle. Furthermore, we can pick an arbitrary reference point P_0 on the curve and define the *arc length s* associated with any other point P as the length of the curve arc separating P_0 and P. Although the arc length depends on the choice of P_0, its differential does not (moving P_0 along the curve amounts to adding a constant to the arc length), and it is often convenient to parameterize a curve by its arc length, where some unspecified choice of origin P_0 is assumed. In particular, the tangent vector at the point P is the unit velocity

$\boldsymbol{t} = dP/ds$. Reversing s also reverses $\boldsymbol{t}$. It can be shown that the acceleration d^2P/ds^2, the curvature κ, and the (principal) normal $\boldsymbol{n}$ are related by

$$\frac{d^2P}{ds^2} = \frac{d\boldsymbol{t}}{ds} = \kappa\boldsymbol{n}.$$

Note that κ and $\boldsymbol{n}$ are both independent of the curve orientation (the negative signs introduced by reversing the direction of traversal of the curve cancel during differentiation), and the curvature is the magnitude of the acceleration. The binormal vector can be defined as $\boldsymbol{b} = \boldsymbol{t} \times \boldsymbol{n}$, and, like $\boldsymbol{t}$, it depends on the orientation chosen for the curve. In general, it is easy to show that

$$\frac{d}{ds}\begin{pmatrix}\boldsymbol{t}\\ \boldsymbol{n}\\ \boldsymbol{b}\end{pmatrix} = \begin{pmatrix}0 & \kappa & 0\\ -\kappa & 0 & \tau\\ 0 & -\tau & 0\end{pmatrix}\begin{pmatrix}\boldsymbol{t}\\ \boldsymbol{n}\\ \boldsymbol{b}\end{pmatrix},$$

where τ denotes the torsion in P. Unlike the curvature, the torsion may be positive, negative or zero for a general space curve. Its sign depends on the direction of traversal chosen for the curve, and it has a geometric meaning: in general, a curve crosses its osculating plane at every point with non-zero torsion, and it emerges on the positive side of that plane (i.e., the same side as the binormal tangent) when the torsion is positive, and on the negative side otherwise. The torsion is of course identically zero for planar curves.

4.1.2 Surfaces

Most of the discussion of the local characteristics of plane and space curves can be generalized in a simple manner to surfaces. Consider a point P on the surface S and all the curves passing through P and lying on S. It can be shown that the tangents to these curves lie in the same plane Π, appropriately called the *tangent plane* in P (Figure 4.11(a)). The line N passing through P and perpendicular to Π is called the *normal line* to P in S, and the surface can be oriented (locally) by picking a sense for a unit *normal vector* along N (unlike curves, surfaces admit a single normal but an infinity of tangents at every point). The surface bounding a solid admits a canonical orientation defined by letting the normal vectors locally point toward the outside of the solid.[2]

Intersecting a surface with the planes that contain the normal in P yields a one-parameter family of planar curves, called *normal sections* (Figure 4.11(b)). These curves are in general regular in P, or they may exhibit an inflection there. The curvature of a normal section is called the *normal curvature* of the surface in the

---

[2] Of course the reverse orientation, where, as Koenderink [**?**, p. 137] puts it, “the normal vector points into the ‘material’ of the blob like the arrows in General Custer’s hat”, is just as valid. The main point is that either choice yields a coherent global orientation of the surface. Certain pathological surfaces (e.g., Möbius strips) do not admit a global orientation, but they do not bound solids.

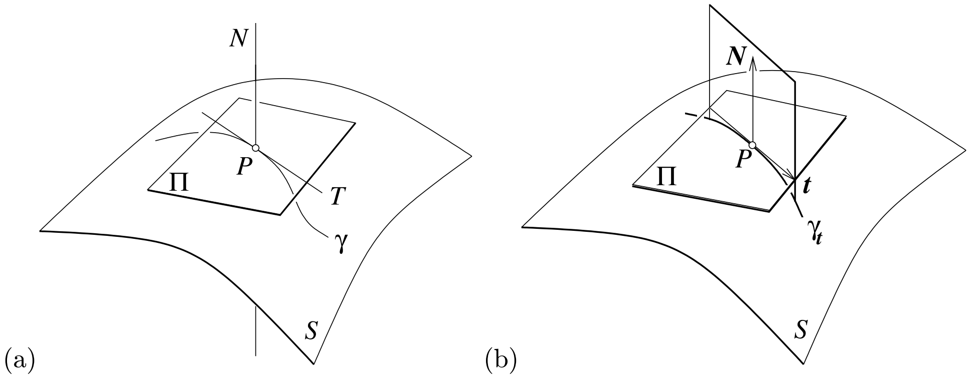


Figure 4.11. Tangent plane and normal sections: (a) the tangent plane Π and the associated normal line N at a point P of a surface; γ is a surface curve passing through P, and its tangent line T lies in Π; (b) the intersection of the surface S with the plane spanned by the normal vector $\boldsymbol{N}$ and the tangent vector $\boldsymbol{t}$ forms a normal section $\gamma_{\boldsymbol{t}}$ of S.

associated tangent direction. By convention, we will choose a positive sign for the normal curvature when the normal section lies (locally) on the same side of the tangent plane as the inward-pointing surface normal, and a negative sign when it lies on the other side. The normal curvature is of course zero when P is an inflection of the corresponding normal section.

With this convention, we can record the normal curvature as the sectioning plane rotates about the surface normal. It will (in general) assume its maximum value κ_1 in a definite direction of the tangent plane, and reach its minimum value κ_2 in a second definite direction. These two directions are called the *principal directions* in P and it can be shown that, unless the normal curvature is constant over all possible orientations, they are orthogonal to each other (see exercises). The *principal curvatures* κ_1 and κ_2 and the associated directions define the best local quadratic approximation of the surface: in particular, we can set up a coordinate system in P with x and y axes along the principal directions and z axis along the outward-pointing normal; the surface can be described (up to second order) in this frame by the paraboloid $z = -1/2(\kappa_1 x^2 + \kappa_2 y^2))$.

The neighborhood of a surface point can locally take three different shapes, depending on the sign of the principal curvatures (Figure 4.12). A point P where both curvatures have the same sign is said to be *elliptic*, and the surface in its vicinity is cup-shaped (Figure 4.12(a)): it does not cross its tangent plane and looks like the surface of an egg (positive curvatures) or the inside surface of its broken shell (negative curvatures). We say that P is convex in the former case and concave in the latter one. When the principal curvatures have opposite signs, we have a *hyperbolic* point. The surface is locally saddle-shaped and crosses its tangent plane along two curves (Figure 4.12(b)). The corresponding normal sections have an inflection in P and their tangents are called the *asymptotic directions* of the surface in P. They are

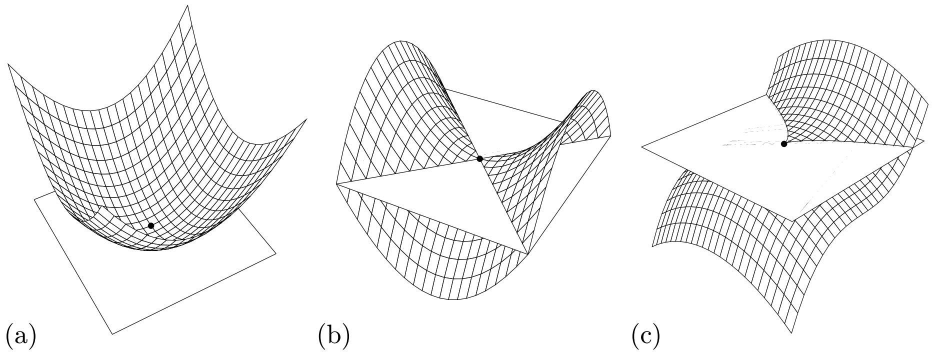


Figure 4.12. Local shape of a surface: (a) an elliptic point, (b) a hyperbolic point, and (c) a parabolic point (there are actually two distinct kinds of parabolic points [**?**]; we will come back to those in Chapter 22).

bisected by the principal directions. The elliptic and hyperbolic points form patches on a surface. These areas are in general separated by curves formed by *parabolic* points where one of the principal curvature vanishes. The corresponding principal direction is also an asymptotic direction, and the intersection of the surface and its tangent plane has (in general) a cusp in that direction (Figure 4.12(c))

Naturally, we can define the Gaussian image of a surface by mapping every point onto the place where the associated unit normal pierces the unit sphere. In the case of plane curves, the Gauss map is one-to-one in the neighborhood of regular points, but the direction of traversal of the Gaussian image reverses in the vicinity of certain singularities. Likewise, it can be shown that the Gauss map is one-to-one in the neighborhood of elliptic or hyperbolic points. The orientation of a small closed curve centered at an elliptic point is preserved by the Gauss map, but the orientation of a curve centered at a hyperbolic point is reversed (Figure 4.13).

The situation is a bit more complex at a parabolic point: in this case, any small neighborhood will contain points with parallel normals, indicating a double covering of the sphere near the parabolic point (Figure 4.13): we say that the Gaussian image *folds* along the parabolic curve. Note the similarity with inflections of planar curves.

Let us now consider a surface curve γ passing through P, parameterized by its arc length s in the neighborhood of P. Since the restriction of the surface normal to γ has constant (unit) length, its derivative with respect to s lies in the tangent plane in P. It is easy to show that the value of this derivative only depends on the unit tangent $\boldsymbol{t}$ to γ and not on γ itself. Thus we can define a mapping $d\boldsymbol{N}$ that associates with each unit vector $\boldsymbol{t}$ in the tangent plane in P the corresponding derivative of the surface normal (Figure 4.14). Using the convention $d\boldsymbol{N}(\lambda\boldsymbol{t}) \overset{\text{def}}{=} \lambda d\boldsymbol{N}(\boldsymbol{t})$ when $\lambda \neq 1$, we can extend $d\boldsymbol{N}$ to a linear mapping defined over the whole tangent plane and called the *differential of the Gauss map* in P.

The *second fundamental form* in P is the bilinear form that associates with any

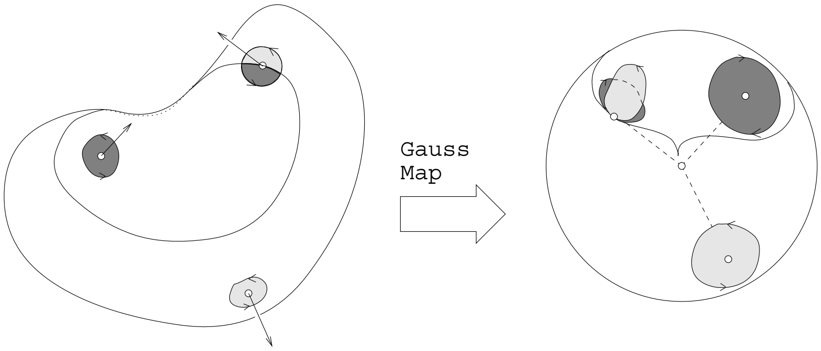


Figure 4.13. Left: a surface in the shape of a kidney bean. It is formed of a convex area, a hyperbolic region, and the parabolic curve separating them. Right: the corresponding Gaussian image. Darkly shaded areas indicate hyperbolic areas, lightly shaded ones indicate elliptic ones. Note that the bean is not convex but does not have any concavity.

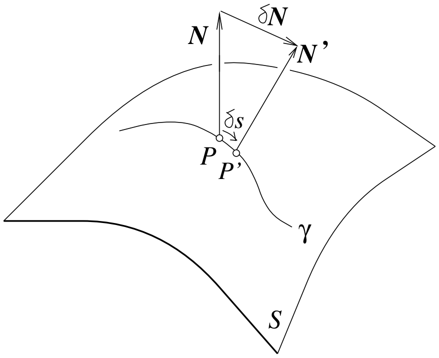


Figure 4.14. The directional derivative of the surface normal: if P and P' are nearby points on the curve γ, and $\boldsymbol{N}$ and $\boldsymbol{N}'$ denote the associated surface normals, with $\delta\boldsymbol{N} = \boldsymbol{N}' - \boldsymbol{N}$, the derivative is defined as the limit of $\delta\boldsymbol{N}/\delta s$ as the length δs of the curve arc separating P and P' tends toward zero.

two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ lying in the tangent plane the quantity[3]

$$\mathrm{II}(\boldsymbol{u},\boldsymbol{v}) \stackrel{\text{def}}{=} \boldsymbol{u}\cdot d\boldsymbol{N}(\boldsymbol{v}).$$

---

[3] The second fundamental form is sometimes defined as $\mathrm{II}(\boldsymbol{u},\boldsymbol{v}) = -\boldsymbol{u}\cdot d\boldsymbol{N}(\boldsymbol{v})$ (see, for example [**?**; **?**]). Our definition allows us to assign positive normal curvatures to the surfaces bounding convex solids with outward-pointing normals.

Application: The shape of specularities

Specularities offer hints about the colour of the illuminant, which we shall exploit in Chapter **??**. They also offer cues to the local geometry of a surface. Understanding these cues is a simple exercise in differential geometry that will serve to illustrate the concepts introduced in this chapter. We consider a smooth specular surface and assume that the radiance reflected in the direction $\boldsymbol{V}$ is a function of $\boldsymbol{V} \cdot \boldsymbol{P}$, where $\boldsymbol{P}$ is the specular direction. We expect the specularity to be small and isolated, so we can assume that the source direction $\boldsymbol{S}$ and the viewing direction $\boldsymbol{V}$ are constant over its extent. Let us further assume that the specularity can be defined by a threshold on the specular energy, i.e., $\boldsymbol{V} \cdot \boldsymbol{P} \geq 1 - \varepsilon$ for some constant ε, denote by $\boldsymbol{N}$ the unit surface normal, and define the **half-angle direction** as $\boldsymbol{H} = (\boldsymbol{S} + \boldsymbol{V})/2$ (Figure 4.15(left)). Using the fact that the vectors $\boldsymbol{S}$, $\boldsymbol{V}$ and $\boldsymbol{P}$ have unit length and a whit of plane geometry, it can easily be shown that the boundary of the specularity is defined by (see exercises):

$$1 - \varepsilon = \boldsymbol{V} \cdot \boldsymbol{P} = 2\frac{(\boldsymbol{H} \cdot \boldsymbol{N})^2}{(\boldsymbol{H} \cdot \boldsymbol{H})} - 1. \tag{4.1.1}$$

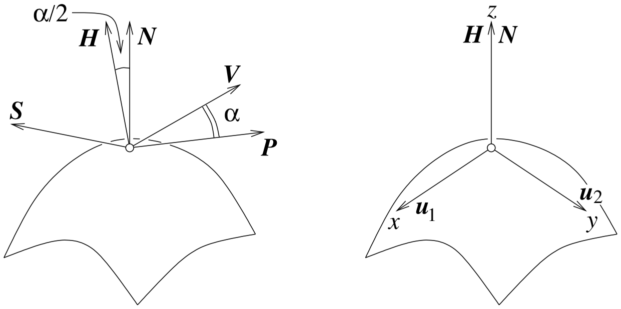


Figure 4.15. A specular surface viewed by a distant observer. We establish a coordinate system at the brightest point of the specularity (where the half-angle direction is equal to the normal) and orient the system using the normal and principal directions.

Because the specularity is small, the second-order structure of the surface will allow us to characterize the shape of its boundary as follows: there is some point on the surface inside the specularity (in fact, the brightest point) where $\boldsymbol{H}$ is parallel to $\boldsymbol{N}$. We set up a coordinate system at this point, oriented so that the z-axis lies along $\boldsymbol{N}$ and the x- and y-axes lie along the principal directions $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ (Figure 4.15(right)). As noted earlier, the surface can be represented up to second order as $z = -1/2(\kappa_1 x^2 + \kappa_2 y^2)$ in this frame, where κ_1 and κ_2 are the principal curvatures. Now, let us define a *parametric surface* as a differentiable mapping $\boldsymbol{x} : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ associating with any couple $(u, v) \in U$ the coordinate vector $(x, y, z)^T$ of a point in some fixed coordinate system. It is easily shown (see exercises) that the normal to a parametric surface is along the vector $\partial\boldsymbol{x}/\partial u \times \partial\boldsymbol{x}/\partial v$. Our second-order surface model is a parametric surface parameterized by x and y, thus its unit surface normal is defined in the corresponding frame by

$$\boldsymbol{N}(x, y) = \frac{1}{\sqrt{1 + \kappa_1^2 x^2 + \kappa_2^2 y^2}} \begin{pmatrix} \kappa_1 x \\ \kappa_2 y \\ 1 \end{pmatrix},$$

and $\boldsymbol{H} = (0,0,1)^T$. Since $\boldsymbol{H}$ is a constant, we can rewrite (4.1.1) as $\kappa_1^2 x^2 + \kappa_2^2 y^2 = \zeta$ where ζ is a constant depending on ε. In particular, the shape of the specularity on the surface contains information about the second fundamental form. The specularity is an ellipse, with major and minor axes along the principal directions, and eccentricity equal to the ratio of the principal curvatures. Unfortunately, the shape of the specularity on the surface is not, in general, directly observable, so this property can only be exploited when a fair amount about the viewing and illumination setup is known [**?**]. While we cannot get much out of the shape of the specularity in the image, it is possible to tell a convex surface from a concave one by watching how a specularity moves as the view changes (you can convince yourself of this with the aid of a spoon). Let us consider a point source at infinity and assume that the specular lobe is very narrow so the viewing direction and the specular direction coincide. Initially, the specular direction is $\boldsymbol{V}$ and the specularity is at the surface point P; after a small eye motion, $\boldsymbol{V}$ changes to $\boldsymbol{V}'$ while the specularity moves to the close-by point P' (Figure 4.16).

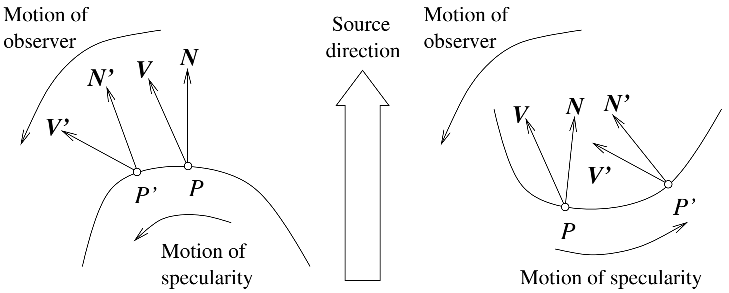


Figure 4.16. Specularities on convex and concave surfaces behave differently when the view changes.

The quantity of interest is $\delta a = (\boldsymbol{V}' - \boldsymbol{V}) \cdot \boldsymbol{t}$, where $\boldsymbol{t} = \frac{1}{\delta s} \overrightarrow{PP'}$ is tangent to the surface, and δs is the (small) distance between P and P': if δa is positive, then the specularity moves in the direction of the view (back of the spoon), and if it is negative, the specularity moves against the direction of the view (bowl of the spoon). By construction, we have $\boldsymbol{V} = 2(\boldsymbol{S} \cdot \boldsymbol{N})\boldsymbol{N} - \boldsymbol{S}$, and

$$\begin{aligned} \boldsymbol{V}' &= 2(\boldsymbol{S} \cdot \boldsymbol{N}')\boldsymbol{N}' - \boldsymbol{S} = 2(\boldsymbol{S} \cdot (\boldsymbol{N} + \delta\boldsymbol{N}))(\boldsymbol{N} + \delta\boldsymbol{N}) - \boldsymbol{S} \\ &= \boldsymbol{V} + 2(\boldsymbol{S} \cdot \delta\boldsymbol{N})\boldsymbol{N} + 2(\boldsymbol{S} \cdot \delta\boldsymbol{N})\delta\boldsymbol{N} + 2(\boldsymbol{S} \cdot \boldsymbol{N})\delta\boldsymbol{N}, \end{aligned}$$

where $\delta\boldsymbol{N} \overset{\text{def}}{=} \boldsymbol{N}' - \boldsymbol{N} = \delta s \, d\boldsymbol{N}(\boldsymbol{t})$. Since $\boldsymbol{t}$ is tangent to the surface in P, ignoring second-order terms yields

$$\delta a = 2\delta s \, (\boldsymbol{S} \cdot \boldsymbol{N}) \mathrm{II}(\boldsymbol{t}, \boldsymbol{t}).$$

Thus, for a concave surface, the specularity always moves against the view and for a convex surface it always moves with the view. The effect is clearly visible if you move a spoon back and forth and look at the images of light sources (of course you could also tell you are looking into the bowl of the spoon by just noticing you see yourself upside-down there). Things are more complex with hyperbolic surfaces; the specularity may move with the view, against the view, or perpendicular to the view (when $\boldsymbol{t}$ is an asymptotic direction).

Since II is easily shown to be symmetric, i.e., $\mathrm{II}(\boldsymbol{u}, \boldsymbol{v}) = \mathrm{II}(\boldsymbol{v}, \boldsymbol{u})$, the mapping that associates with any tangent vector $\boldsymbol{u}$ the quantity $\mathrm{II}(\boldsymbol{u}, \boldsymbol{u})$ is a quadratic form. In turn, this quadratic form is intimately related to the curvature of the surface curves passing through P. Indeed, note that the tangent $\boldsymbol{t}$ to a surface curve is everywhere orthogonal to the surface normal $\boldsymbol{N}$. Differentiating the dot product of these two vectors with respect to the curve arc length yields

$$\kappa \boldsymbol{n} \cdot \boldsymbol{N} + \boldsymbol{t} \cdot d\boldsymbol{N}(\boldsymbol{t}) = 0,$$

where $\boldsymbol{n}$ denotes the principal normal to the curve, and κ denotes its curvature. This can be rewritten as

$$\mathrm{II}(\boldsymbol{t}, \boldsymbol{t}) = -\kappa \cos\phi. \tag{4.1.2}$$

where ϕ is the angle between the surface and curve normals. For normal sections, we have $\boldsymbol{n} = \mp \boldsymbol{N}$, and it follows that the normal curvature in some direction $\boldsymbol{t}$ is

$$\kappa_{\boldsymbol{t}} = \mathrm{II}(\boldsymbol{t}, \boldsymbol{t}),$$

where, as before, we use the convention that the normal curvature is positive when the principal normal to the curve and the surface normal point in opposite directions.

In addition, (4.1.2) shows that the curvature κ of a surface curve whose principal normal makes an angle ϕ with the surface normal is related to the normal curvature $\kappa_{\boldsymbol{t}}$ in the direction of its tangent $\boldsymbol{t}$ by $\kappa \cos\phi = -\kappa_{\boldsymbol{t}}$. This is known as Meusnier's theorem (Figure 4.17).

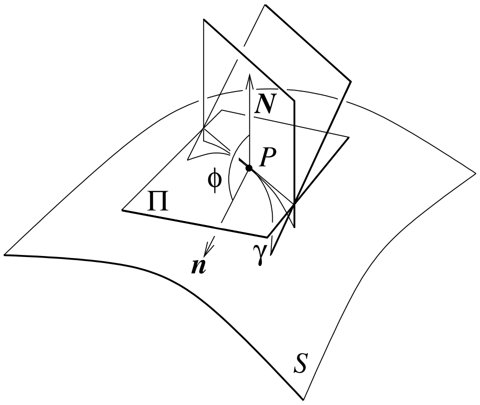


Figure 4.17. Meusnier's theorem.

It turns out that the principal directions are the eigenvectors of the linear map $d\boldsymbol{N}$, and the principal curvatures are the associated eigenvalues. The determinant K of this map is called the Gaussian curvature, and it is equal to the product of the principal curvatures. Thus, the sign of the Gaussian curvature determines the local shape of the surface: a point is elliptic when $K > 0$, hyperbolic when $K < 0$, and parabolic when $K = 0$. If δA is the area of a small patch centered in P on a surface S, and $\delta A'$ is the area of the corresponding patch of the Gaussian image of S, it can also be shown that the Gaussian curvature is the limit of the (signed) ratio $\delta A'/\delta A$ as both areas approach zero (by convention, the ratio is chosen to be positive when the boundaries of both small patches have the same orientation, and negative otherwise, see Figure 4.13). Note again the strong similarity with the corresponding concepts (Gaussian image and plain curvature) in the context of planar curves.

4.2 Contour Geometry

Before studying the geometry of surface outlines, let us pose for a minute and examine the relationship between the local shape of a twisted curve Γ and that of its orthographic projection γ onto some plane Π (Figure 4.18). Let us denote by α the angle between the tangent to Γ and the plane Π, and by β the angle betweem the principal normal to Γ and Π. These two angles completely define the local orientation of the curve relative to the image plane.

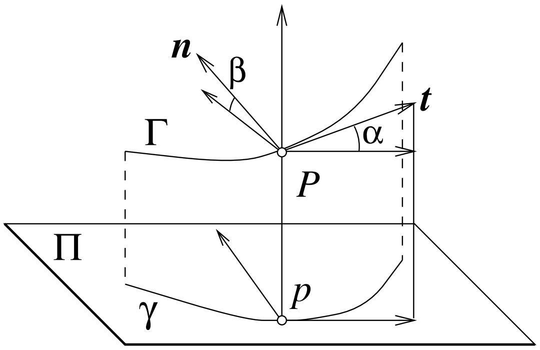


Figure 4.18. A space curve and its projection Note that the tangent to γ is the projection of the tangent to Γ (think for example of the tangent as the velocity of a particle traveling along the curve). The normal to γ is *not*, in general, the projection of the normal to Γ.

If κ denotes the curvature of Γ at some point, and κ_a denotes its *apparent curvature*, i.e., the curvature of γ at the corresponding image point, it is easy to show analytically (see exercises) that

$$\kappa_a = \kappa \frac{\cos\beta}{\cos^2\alpha}. \tag{4.2.1}$$

In particular, when the viewing direction is in the osculating plane ($\cos\beta = 0$), the apparent curvature κ_a vanishes and the image of the curve acquires an inflection.

When, on the other hand, the viewing direction is tangent to the curve ($\cos\alpha = 0$), κ_a becomes infinite and the projection acquires a cusp.

These two properties of the projections of space curves are well known and mentioned in all differential geometry textbooks. Is it possible to relate in a similar fashion the local shape of the surface bounding a solid object to the shape of its image contour? The answer is a resounding "Yes!", as shown by Koenderink [**?**] in his delightful paper "What does the occluding contour tell us about solid shape?," and we present in this section a few elementary properties of image contours, before stating and proving the main theorem of Koenderink's paper, and concluding by discussing some of its implications.

4.2.1 The Occluding Contour and the Image Contour

As noted earlier, the image of a solid bounded by a smooth surface is itself bounded by an image curve, called the contour, silhouette or outline of this solid. This curve is the intersection of the retina with a viewing cone whose apex coincides with the pinhole and whose surface grazes the object along a second curve, called the occluding contour, or rim (Figure 4.3).

We will assume orthographic projection in the rest of this section. In this case, the pinhole moves to infinity and the viewing cone becomes a cylinder whose generators are parallel to the (fixed) viewing direction. The surface normal is constant along each one of these generators, and it is parallel to the image plane (Figure 4.19). The tangent plane at a point on the occluding contour projects onto the tangent to the image contour, and it follows that the normal to this contour is equal to the surface normal at the corresponding point of the occluding contour.

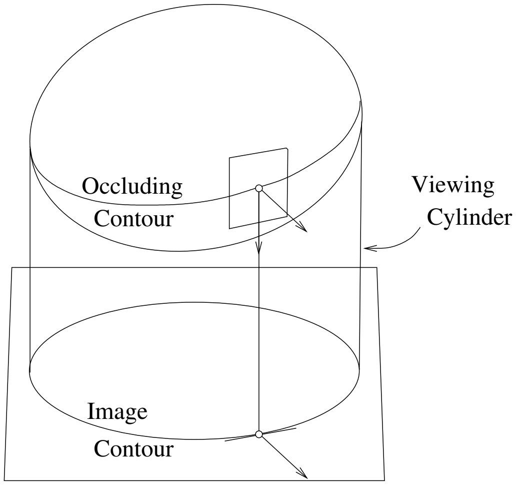


Figure 4.19. Occluding boundaries under orthographic projection.

It is important to note that the viewing direction $\boldsymbol{v}$ is *not*, in general, perpendicular to the occluding contour tangent $\boldsymbol{t}$ (as noted by Nalwa [?] for example, the occluding contour of a tilted cylinder is parallel to its axis and not to the image plane). Instead, it can be shown that these two directions are *conjugated*, an extremely important property of the occluding contour.

4.2.2 The Cusps and Inflections of the Image Contour

Two directions $\boldsymbol{u}$ and $\boldsymbol{v}$ in the tangent plane are said to be conjugated when $\mathrm{II}(\boldsymbol{u}, \boldsymbol{v}) = 0$. For example, the principal directions are conjugated since they are orthogonal eigenvectors of $d\boldsymbol{N}$, and asymptotic directions are self-conjugated.

It is easy to show that the tangent $\boldsymbol{t}$ to the occluding contour is always conjugated to the corresponding projection direction $\boldsymbol{v}$: indeed, $\boldsymbol{v}$ is tangent to the surface at every point of the occluding contour, and differentiating the identity $\boldsymbol{N} \cdot \boldsymbol{v} = 0$ with respect to the arc length of this curve yields

$$0 = (\frac{d}{ds}\boldsymbol{N}) \cdot \boldsymbol{v} = d\boldsymbol{N}(\boldsymbol{t}) \cdot \boldsymbol{v} = \mathrm{II}(\boldsymbol{t}, \boldsymbol{v}).$$

Let us now consider a hyperbolic point P_0 and project the surface onto a plane perpendicular to one of its asymptotic directions. Since asymptotic directions are self-conjugated, the occluding contour in P_0 must run along this direction. As shown by (4.2.1), the curvature of the contour must be infinite in that case, and the contour acquires a cusp of the first kind.

We will state in a moment a theorem by Koenderink [?] that provides a quantitative relationship between the curvature of the image contour and the Gaussian curvature of the surface. In the mean time, we will prove (informally) a weaker, but still remarkable result: *under orthographic projection, the inflections of the contour are images of parabolic points* (Figure 4.20).

First note that, under orthographic projection, the surface normal at a point on the occluding contour is the same as the normal at the corresponding point of the image contour. Since the Gaussian image of a surface folds at a parabolic point, it follows that the Gaussian image of the image contour must reverse direction at such a point. As shown earlier, the Gaussian image of a planar curve reverses at its inflections and cusps of the second kind. It is easily shown that the latter singularity does not occur for a general viewpoint, which proves the result.

In summary, the occluding contour is formed by points where the viewing direction $\boldsymbol{v}$ is tangent to the surface (the fold points mentioned in the introduction). Occasionally, it becomes tangent to $\boldsymbol{v}$ or crosses a parabolic line, and cusps (of the first kind) or inflections appear on the contour accordingly. Unlike the curves mentioned so far, the image contour may also cross itself (transversally) when two distinct branches of the occluding contour project onto the same image point, forming a T-junction (Figure 4.3). For general viewpoints, these are the only possibilities: there is no cusp of the second kind, nor any tangential self-intersection for example.

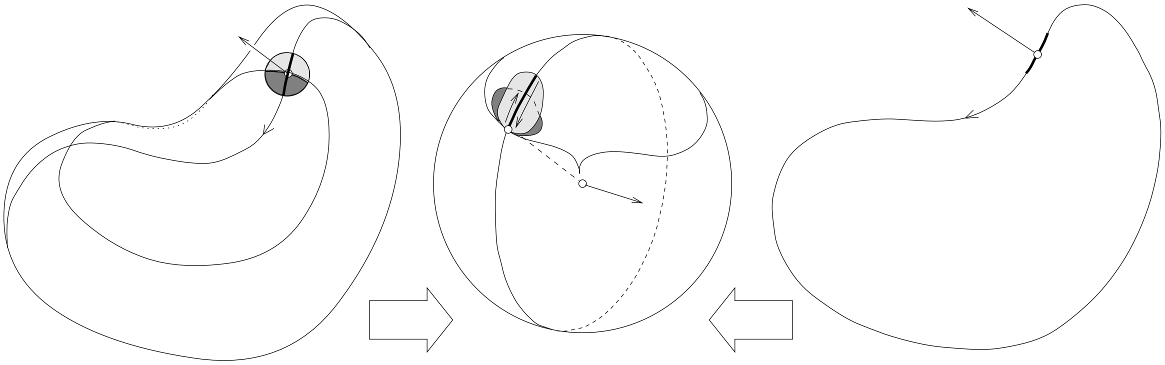


Figure 4.20. The inflections of the contour are images of parabolic points: the left side of this diagram shows the bean-shaped surface with an occluding contour overlaid, and its right side shows the corresponding image contour. The Gaussian image folds at the parabolic point, and so does its restriction to the great circle formed by the image of the occluding and image contours.

We will come back to the study of exceptional viewpoints and the corresponding contour singularities in a latter chapter.

4.2.3 Koenderink's Theorem

Let us now state the theorem by Koenderink [**?**] that has already been mentioned several times. We assume as before orthographic projection, consider a point P on the occluding contour of a surface S, and denote by p its image on the contour.

Theorem 1: *The Gaussian curvature K of S in P and the contour curvature κ_c in p are related by*

$$K = \kappa_c \kappa_r,$$

where κ_r denotes the curvature of the radial curve *formed by the intersection of S with the plane defined by the normal to S in P and the projection direction (Figure 4.21).*

This remarkably simple relation has several important corollaries. Note first that κ_r remains positive (or zero) along the occluding contour since the projection ray locally lies inside the imaged object at any point where $\kappa_r < 0$. It follows that κ_c will be positive when the Gaussian curvature is positive, and negative otherwise. In particular, the theorem shows that convexities of the contour corresponds to elliptic points of the surface, while contour concavities correspond to hyperbolic points and contour inflections correspond to parabolic points.

Among elliptic surface points, it is clear that concave points never appear on the occluding contour of an opaque solid since their tangent plane lies (locally) completely inside this solid (Figure 4.21(b)). Thus convexities of the contour also

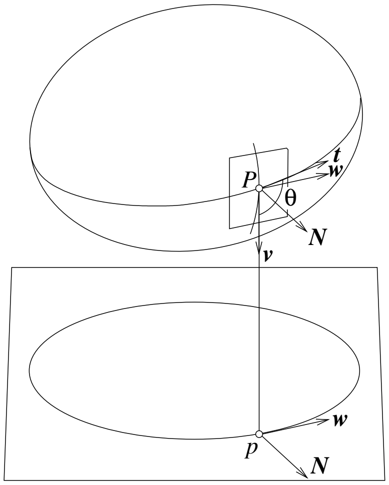


Figure 4.21. The relationship between occluding contour and image contour: the viewing direction $\boldsymbol{v}$ and the occluding contour tangent $\boldsymbol{t}$ are conjugated, and the radial curvature is always nonnegative at a visible point of the contour for opaque solids.

correspond to convexities of the surface. Likewise, we saw earlier that the contour cusps when the viewing direction is an asymptotic direction at a hyperbolic point. In the case of an opaque object, this means that concave arcs of the contour may terminate at such a cusp, where a branch of the contour becomes occluded. Thus we see that Koenderink's theorem strengthens and refines the earlier characterization of the geometric properties of image contours.

Let us now prove the theorem. It is related to a general property of conjugated directions: if $\kappa_{\boldsymbol{u}}$ and $\kappa_{\boldsymbol{v}}$ denote the normal curvatures in conjugated directions $\boldsymbol{u}$ and $\boldsymbol{v}$, and K denotes the Gaussian curvature, then

$$K \sin^2 \theta = \kappa_{\boldsymbol{u}} \kappa_{\boldsymbol{v}}, \tag{4.2.2}$$

where θ is the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$. This relation is easy to prove by using the fact that the matrix associated with the second fundamental form is diagonal in the basis of the tangent plane formed by conjugated directions (see exercises). It is obviously satisfied for principal directions ($\theta = \pi/2$) and asymptotic ones ($\theta = 0$).

In the context of Koenderink's theorem, we obtain

$$K \sin^2 \theta = \kappa_r \kappa_{\boldsymbol{t}},$$

where $\kappa_{\boldsymbol{t}}$ denotes the normal curvature of the surface along the occluding contour direction $\boldsymbol{t}$ (which is of course different from the actual curvature of the occluding contour).

To complete the proof of the theorem, we use another general property of surfaces: the apparent curvature of any surface curve with tangent $\boldsymbol{t}$ is

$$\kappa_a = \frac{\kappa_{\boldsymbol{t}}}{\cos^2 \alpha}, \tag{4.2.3}$$

where α denotes as before the angle between $\boldsymbol{t}$ and the image plane. Indeed, we can use Meusnier's theorem to rewrite (4.2.1) as

$$\kappa_a = \kappa \frac{\cos \beta}{\cos^2 \alpha} = \frac{\kappa_{\boldsymbol{t}} \cos \beta}{-\cos^2 \alpha \cos \phi} = \frac{\kappa_{\boldsymbol{t}}}{\cos^2 \alpha}$$

since in this case $\phi = \beta + \pi$. In other words, the apparent cuvature of any surface curve is obtained by dividing the associated normal curvature by the square of the cosine of the angle between its tangent and the image plane. Applying this result to the occluding contour yields

$$\kappa_c = \frac{\kappa_{\boldsymbol{t}}}{\sin^2 \theta} \tag{4.2.4}$$

since $\alpha = \theta - \pi/2$. Substituting (4.2.4) into (4.2.2) concludes the proof of the theorem.

4.3 Notes

There is a rich literature on the three-dimensional interpretation of line-drawings, including the seminal work by Huffman [**?**], Clowes [**?**] and Waltz [**?**], that uses the relatively small set of possible junction labels associated with typical scenes to control the combinatorial explosion of the line-labeling process. More quantitative approaches have also been proposed (see, for example [**?**; **?**]), and the technique based on linear programming proposed by Sugihara [**?**] is often considered as the ultimate achievement in this field. The relationship between contour and surface orientation is studied in [**?**]. See also the work of Malik [**?**] for an extension of the junction catalogue to piecewise-smooth surfaces.

There is of course a large number of excellent textbooks on differential geometry, including the very accessible presentations found in the books of do Carmo [**?**] and Struik [**?**]. Our presentation is closer in spirit (if not in elegance) to the descriptive introduction to differential geometry found in Hilbert's and Cohn-Vossen's wonderful book "Geometry and the Imagination" [**?**].

It was not always recognized that the image contour carries vital information about surface shape: see [**?**; **?**] for arguments to the contrary. The theorem proven in this chapter clarified the situation and appeared first in [**?**]. Our proof is different from the original one, but it is close in spirit to the proof given in [**?**], which is based on Blaschke's dual version of Euler's theorem. Our choice here was motivated by our reluctance to use any formulas that require setting a particular coordinate system. Alternate proofs for various kinds of projection geometries can be found in [**?**; **?**; **?**; **?**; **?**].

4.4 Assignments

Exercises

1. Prove that the curvature κ of a planar curve in a point P is the inverse of the radius of curvature r at this point.

   Hint: use the fact that $\sin u \approx u$ for small angles.

2. Prove that, unless the normal curvature is constant over all possible directions, the principal directions are orthogonal to each other.

3. Prove that the second fundamental form is bilinear and symmetric.

4. Consider a fixed coordinate system, and a parametric surface $\boldsymbol{x} : U \subset \mathbb{R}^2 \to \mathbb{R}^3$. Show that the normal to a parametric surface is parallel to the vector $\partial \boldsymbol{x}/\partial u \times \partial \boldsymbol{x}/\partial v$.

   Hint: consider the two surface curves $u = u_0$ and $v = v_0$ passing through the point $\boldsymbol{x}(u_0, v_0)$.

5. Consider a specular surface observed by a distant observer. Denote by $\boldsymbol{V}$ the viewing direction, $\boldsymbol{P}$ the specular direction, and $\boldsymbol{S}$ the light source direction, all unit vectors. Define the **half-angle direction** $\boldsymbol{H} = (\boldsymbol{S} + \boldsymbol{V})/2$. Show that

   $$\boldsymbol{V} \cdot \boldsymbol{P} = 2\frac{(\boldsymbol{H} \cdot \boldsymbol{N})^2}{(\boldsymbol{H} \cdot \boldsymbol{H})} - 1,$$

   where $\boldsymbol{N}$ is the unit surface normal.

6. Let us denote by α the angle between the tangent to Γ and the plane Π, by β the angle betweem the principal normal to Γ and Π, and by κ the curvature of Γ at some point. Prove that if κ_a denotes its *apparent curvature*, i.e., the curvature of γ at the corresponding point, then

   $$\kappa_a = \kappa \frac{\cos \beta}{\cos^2 \alpha}.$$

   (Note: this is a variant of a result by Koenderink [**?**, p. 191], who uses a different notation.)

7. Let $\kappa_{\boldsymbol{u}}$ and $\kappa_{\boldsymbol{v}}$ denote the normal curvatures in conjugated directions $\boldsymbol{u}$ and $\boldsymbol{v}$ at a point P, and let K denote the Gaussian curvature, prove that:

   $$K \sin^2 \theta = \kappa_{\boldsymbol{u}} \kappa_{\boldsymbol{v}},$$

   where θ is the angle between the $\boldsymbol{u}$ and $\boldsymbol{v}$.

PROOF: Note that the second fundamental form can be written in the basis $(\boldsymbol{u}, \boldsymbol{v})$ as

$$\mathrm{II}(x\boldsymbol{u}+y\boldsymbol{v}, x\boldsymbol{u}+y\boldsymbol{v}) = (x, y)\begin{pmatrix} \mathrm{II}(\boldsymbol{u},\boldsymbol{u}) & \mathrm{II}(\boldsymbol{u},\boldsymbol{v}) \\ \mathrm{II}(\boldsymbol{u},\boldsymbol{v}) & \mathrm{II}(\boldsymbol{v},\boldsymbol{v}) \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = (x, y)\begin{pmatrix} \kappa_{\boldsymbol{u}} & 0 \\ 0 & \kappa_{\boldsymbol{v}} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}.$$

In other words, the second fundamental form has a diagonal matrix in the basis $(\boldsymbol{u}, \boldsymbol{v})$. If $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ denote the principal directions, and $x\boldsymbol{u} + y\boldsymbol{v} = x'\boldsymbol{e}_1 + y'\boldsymbol{e}_2$, we have

$$(x', y')\,(\,\boldsymbol{e}_1, \boldsymbol{e}_2\,)^T\,(\,\boldsymbol{u}, \boldsymbol{v}\,)^{-T}\begin{pmatrix} \kappa_{\boldsymbol{u}} & 0 \\ 0 & \kappa_{\boldsymbol{v}} \end{pmatrix}(\,\boldsymbol{u}, \boldsymbol{v}\,)^{-1}\,(\,\boldsymbol{e}_1, \boldsymbol{e}_2\,)\begin{pmatrix} x' \\ y' \end{pmatrix}.$$

Since the Gaussian curvature is the determinant of the differential of the Gauss map, we have therefore

$$K = \frac{\kappa_{\boldsymbol{u}}\kappa_{\boldsymbol{v}}}{|\,(\,\boldsymbol{u}, \boldsymbol{v}\,)\,|^2} = \frac{\kappa_{\boldsymbol{u}}\kappa_{\boldsymbol{v}}}{\sin^2\theta}.$$

■

8. Show that the occluding is a smooth curve that does not intersect itself.

   Hint: use the Gauss map.

Chapter 5

ANALYTICAL IMAGE FEATURES

Chapter **??** laid the geometric foundations of image formation. This chapter uses analytical geometry to quantify more precisely the relationship between a camera, the objects it observes, and the pictures of these objects. We start by briefly recalling elementary notions of analytical Euclidean geometry, including dot and cross products, norms and distances, rigid transformations and homogeneous coordinates: this machinery will allow us in the rest of the book to reason about geometric objects like points, lines and planes, and geometric transformations like rotations, translations and projections in terms of linear algebra constructs such as vectors and matrices. We then introduce the various physical parameters that relate the world and camera coordinate frames, and present as an application various methods for estimating these parameters, a process known as geometric camera calibration. We also present along the way some linear and non-linear least-squares techniques for parameter estimation that will prove useful on several occasions in the rest of the book.

5.1 Elements of Analytical Euclidean Geometry

We assume that the reader has some familiarity with elementary analytical Euclidean geometry and linear algebra. This section will serve to fix the notation used in the book and introduce informally some useful notions such as coordinate systems, homogeneous coordinates, rotation matrices, etc.

Notation. We will use the following notation in the rest of this chapter and throughout the book: points, lines and planes will be denoted by Roman or Greek letters in italic font, e.g., P, Δ or Π. Vectors will be denoted by Roman or Greek bold-italic letters, e.g., $\boldsymbol{v}$, $\boldsymbol{P}$, or $\boldsymbol{\xi}$, although the vector joining two points P and Q will often be denoted by $\overrightarrow{PQ}$. Matrices will be denoted by Roman letters in calligraphic font, e.g., $\mathcal{M}$. The familiar three-dimensional Euclidean space will be denoted by $\mathbb{E}^3$ and the vector space formed by n-tuples of real numbers with the usual laws of addition and multiplication by a scalar will be denoted by $\mathbb{R}^n$. Elements of $\mathbb{R}^n$ will be considered as column vectors or $n \times 1$ matrices, and the transpose of the $m \times n$ matrix $\mathcal{A}$ with coefficients a_{ij} will be the $n \times m$ matrix denoted by $\mathcal{A}^T$ with coefficients a_{ji}.

We will denote the dot product (or inner product) between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as $\boldsymbol{u} \cdot \boldsymbol{v}$. When these two vectors are elements of $\mathbb{R}^n$ given by $\boldsymbol{u} = (u_1, \dots, u_n)^T$ and $\boldsymbol{v} = (v_1, \dots, v_n)$, we have of course

$$\boldsymbol{u} \cdot \boldsymbol{v} = u_1 v_1 + \ldots + u_n v_n,$$

and we will often use the fact that in this case the dot product can be rewritten as a matrix product, i.e., $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v} = \boldsymbol{v}^T \boldsymbol{u}$. We will denote by $|\boldsymbol{v}|^2 = \boldsymbol{v} \cdot \boldsymbol{v}$ the square of the Euclidean norm of the vector $\boldsymbol{v}$, and denote by d the distance function induced by the Euclidean norm in $\mathbb{E}^3$, i.e., $d(P, Q) = |\overrightarrow{PQ}|$.

The symbol "×" will be used to denote the cross product (or outer product) operator that associates with two vectors $\boldsymbol{u} = (u_1, u_2, u_3)^T$ and $\boldsymbol{v} = (v_1, v_2, v_3)^T$ the vector

$$\boldsymbol{u} \times \boldsymbol{v} \stackrel{\text{def}}{=} \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix}.$$

When $\boldsymbol{u}$ has unit norm, the dot product $\boldsymbol{u} \cdot \boldsymbol{v}$ is equal to the (signed) length of the projection of $\boldsymbol{v}$ onto $\boldsymbol{u}$, and two vectors are orthogonal when their dot product is zero. On the other hand, the cross product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\mathbb{R}^3$ is orthogonal to these two vectors, and a necessary and sufficient condition for $\boldsymbol{u}$ and $\boldsymbol{v}$ to have the same direction is that $\boldsymbol{u} \times \boldsymbol{v} = 0$. We will also use the identities

$$\begin{cases} (\boldsymbol{u} \cdot \boldsymbol{v})^2 = |\boldsymbol{u}|^2 |\boldsymbol{v}|^2 \cos^2 \theta, \\ |\boldsymbol{v} \times \boldsymbol{v}|^2 = |\boldsymbol{u}|^2 |\boldsymbol{v}|^2 \sin^2 \theta, \end{cases}$$

where θ denotes the angle between the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

5.1.1 Coordinate Systems and Homogeneous Coordinates

We already used three-dimensional coordinate systems in Chapter ??. Let us introduce them a bit more formally: we assume a fixed system of units, say meters, or inches, so unit length is well defined; picking a point O in $\mathbb{E}^3$ and three unit vectors $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ orthogonal to each other defines an *orthonormal coordinate frame* (F) as the quadruple $(O, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k})$. The point O is the *origin* of the coordinate system (F), and $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ are its *basis vectors*. We will restrict our attention to *right-handed* coordinate systems, such that the vectors $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ can be thought of as being attached to fingers of your right hand, with the thumb pointing up, index pointing straight, and middle finger pointing left as shown in Figure 5.1.[1]

---

[1] This is the traditional way of defining right-handed coordinate systems. One of the authors, who is left-handed, has always found it a bit confusing, and prefers to identify these coordinate systems using the fact that when one looks down the $\boldsymbol{k}$ axis at the $(\boldsymbol{i}, \boldsymbol{j})$ plane, the vector $\boldsymbol{i}$ is mapped onto the vector $\boldsymbol{j}$ by a *counterclockwise* 90° rotation (Figure 5.1). Left-handed coordinate systems correspond to *clockwise* rotations. Left- and right-handed readers alike may find this characterization useful as well.

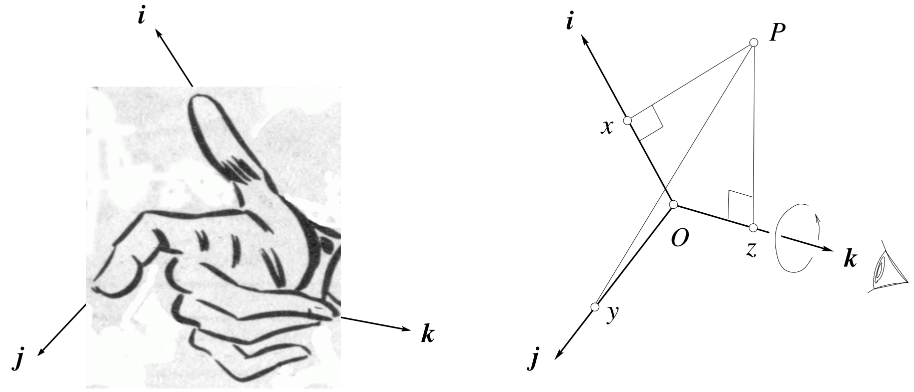


Figure 5.1. A right-handed coordinate system and the Cartesian coordinates x, y, z of a point P.

The Cartesian coordinates x, y and z of a point P in this coordinate frame are defined as the (signed) lengths of the orthogonal projections of the vector $\overrightarrow{OP}$ onto the vectors $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ (Figure 5.1), with

$$\begin{cases} x = \overrightarrow{OP} \cdot \boldsymbol{i} \\ y = \overrightarrow{OP} \cdot \boldsymbol{j} \\ z = \overrightarrow{OP} \cdot \boldsymbol{k} \end{cases} \iff \overrightarrow{OP} = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}.$$

The column vector

$$\boldsymbol{P} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$$

is called the *coordinate vector* of the point P in (F). We can also define the coordinate vector associated with any free vector $\boldsymbol{v}$ by the lengths of its projections onto the basis vectors of (F), and these coordinates are of course independent of the choice of the origin O.

Let us now consider a plane Π, an arbitrary point A in Π and a unit vector $\boldsymbol{n}$ perpendicular to the plane. The points lying in Π are characterized by

$$\overrightarrow{AP} \cdot \boldsymbol{n} = 0.$$

In a coordinate system (F) where the coordinates of the point P are x, y, z and the coordinates of $\boldsymbol{n}$ are a, b and c, this can be rewritten as $\overrightarrow{OP} \cdot \boldsymbol{n} - \overrightarrow{OA} \cdot \boldsymbol{n} = 0$ or

$$ax + by + cz - d = 0, \tag{5.1.1}$$

where $d \overset{\text{def}}{=} \overrightarrow{OA} \cdot \boldsymbol{n}$ is independent of the choice of the point A in Π and is simply the (signed) distance between the origin O and the plane Π (Figure 5.2)

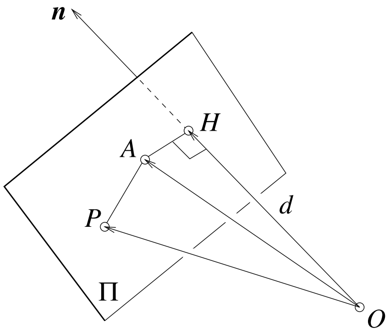


Figure 5.2. The geometric definition of the equation of a plane. The distance d between the origin and the plane is reached at the point H where the normal vector passing through the origin pierces the plane.

At times, it is useful to use *homogeneous coordinates* to represent points, vectors, and planes. We will justify formally their definition later in this book, when we introduce notions of affine and projective geometry, but for the time being, let us just note that (5.1.1) can be rewritten as

$$(a, b, c, -d)\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0$$

or, more concisely, as

$$\boldsymbol{\Pi} \cdot \boldsymbol{P} = 0, \quad \text{where} \quad \boldsymbol{\Pi} \stackrel{\text{def}}{=} \begin{pmatrix} a \\ b \\ c \\ -d \end{pmatrix} \quad \text{and} \quad \boldsymbol{P} \stackrel{\text{def}}{=} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{5.1.2}$$

The vector $\boldsymbol{P}$ is called the *vector of homogenous coordinates* of the point P in the coordinate system (F), and it is simply obtained by adding a fourth coordinate equal to 1 to the ordinary coordinate vector of P. Likewise, the vector $\boldsymbol{\Pi}$ is the vector of homogeneous coordinates of the plane Π in the coordinate frame (F) and (5.1.2) is called the equation of Π in that coordinate system. Note that $\boldsymbol{\Pi}$ is only defined up to scale since multiplying this vector by any nonzero constant does not change the solutions of (5.1.2).

We will use the convention that homogeneous coordinates are only defined up to scale, whether they represent points or planes (this will be established more formally for points later). To go back to the ordinary non-homogenous coordinates of points, one just divides all coordinates by the fourth one. Among other things, homogeneous coordinates will allow us shortly to express changes of coordinate in terms of vectors of matrices, but first, we have to understand how coordinates change between two frames.

5.1.2 Coordinate System Changes and Rigid Transformations

When several different coordinate systems are considered at the same time, it is convenient to follow Craig [**?**] and denote by ${}^F P$ (resp. ${}^F \boldsymbol{v}$) the coordinate vector of the point P (resp. vector $\boldsymbol{v}$) in the frame (F), i.e.,[2]

$$ {}^F P = {}^F \overrightarrow{OP} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Longleftrightarrow \overrightarrow{OP} = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}. $$

Let us now consider two coordinate systems $(A) = (O_A, \boldsymbol{i}_A, \boldsymbol{j}_A, \boldsymbol{k}_A)$ and $(B) = (O_B, \boldsymbol{i}_B, \boldsymbol{j}_B, \boldsymbol{k}_B)$. The rest of this section will allow us to express ${}^B P$ as a function of ${}^A P$. Let us suppose first that the basis vectors of both coordinate systems are parallel to each other, i.e., $\boldsymbol{i}_A = \boldsymbol{i}_B$, $\boldsymbol{j}_A = \boldsymbol{j}_B$ and $\boldsymbol{k}_A = \boldsymbol{k}_B$, but the origins O_A and O_B are distinct (Figure 5.3). We say that the two coordinate systems are separated by a *pure translation*, and we have $\overrightarrow{O_B P} = \overrightarrow{O_B O_A} + \overrightarrow{O_A P}$, thus

$$ {}^B P = {}^A P + {}^B O_A. $$

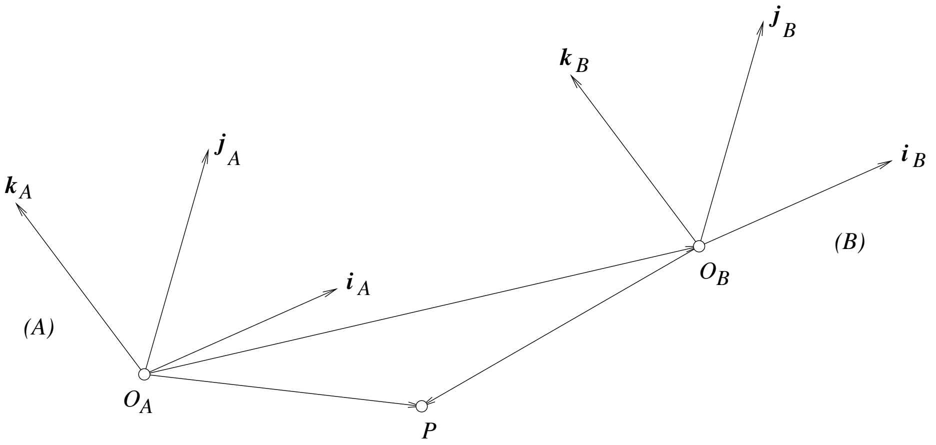


Figure 5.3. Coordinate change between two frames: pure translation.

When the origins of the two frames coincide, i.e., $O_A = O_B = O$, we say that the frames are separated by a *pure rotation* (Figure 5.4). Let us define the *rotation matrix* ${}^B_A\mathcal{R}$ as the 3×3 array of numbers

$$ {}^B_A\mathcal{R} \overset{\text{def}}{=} \begin{pmatrix} \boldsymbol{i}_A \cdot \boldsymbol{i}_B & \boldsymbol{j}_A \cdot \boldsymbol{i}_B & \boldsymbol{k}_A \cdot \boldsymbol{i}_B \\ \boldsymbol{i}_A \cdot \boldsymbol{j}_B & \boldsymbol{j}_A \cdot \boldsymbol{j}_B & \boldsymbol{k}_A \cdot \boldsymbol{j}_B \\ \boldsymbol{i}_A \cdot \boldsymbol{k}_B & \boldsymbol{j}_A \cdot \boldsymbol{k}_B & \boldsymbol{k}_A \cdot \boldsymbol{k}_B \end{pmatrix}. $$

---

[2] The superscripts and subscripts preceding points, vectors and matrices in Craig's notation may appear awkward at first, but the rest of this section should clearly demonstrate their utility. Please stick with us for a little while..

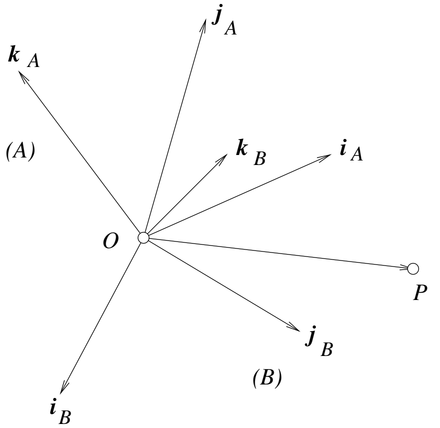


Figure 5.4. Coordinate change between two frames: pure rotation.

Note that the first column of ${}^B_A\mathcal{R}$ is formed by the coordinates of $\boldsymbol{i}_A$ in the basis $(\boldsymbol{i}_B, \boldsymbol{j}_B, \boldsymbol{k}_B)$. Likewise, the third row of this matrix is formed by the coordinates of $\boldsymbol{k}_B$ in the basis $(\boldsymbol{i}_A, \boldsymbol{j}_A, \boldsymbol{k}_A)$, etc. More generally, the matrix ${}^B_A\mathcal{R}$ can be written in a more compact fashion using a combination of three column vectors or three row vectors:

$$ {}^B_A\mathcal{R} = ({}^B\boldsymbol{i}_A \quad {}^B\boldsymbol{j}_A \quad {}^B\boldsymbol{k}_A) = \begin{pmatrix} {}^A\boldsymbol{i}_B{}^T \\ {}^A\boldsymbol{j}_B{}^T \\ {}^A\boldsymbol{k}_B{}^T \end{pmatrix}, $$

and it follows that ${}^A_B\mathcal{R} = {}^B_A\mathcal{R}^T$.

As noted earlier, all these subscripts and superscripts may be somewhat confusing at first. To keep everything straight, it is useful to remember that in a change of coordinates, subscripts refer to the object being described, while superscripts refer to the coordinate system in which the object is described. For example ${}^AP$ refers to the coordinate vector of the point P in the frame (A), ${}^B\boldsymbol{j}_A$ is the coordinate vector of the vector $\boldsymbol{j}_A$ in the frame (B), and ${}^B_A\mathcal{R}$ is the rotation matrix describing the frame (A) in the coordinate system (B).

Let us give an example of pure rotation: suppose that $\boldsymbol{k}_A = \boldsymbol{k}_B = \boldsymbol{k}$, and denote by θ the angle such that the vector $\boldsymbol{i}_B$ is obtained by applying to the vector $\boldsymbol{i}_A$ a counterclockwise rotation of angle θ about $\boldsymbol{k}$ (Figure 5.5). The angle between the vectors $\boldsymbol{j}_A$ and $\boldsymbol{j}_B$ is also θ in this case, and we have

$$ {}^B_A\mathcal{R} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.1.3} $$

Similar formulas can be written when the two coordinate systems are deduced from each other via rotations about the $\boldsymbol{i}_A$ or $\boldsymbol{j}_A$ axes (see exercises). In general,

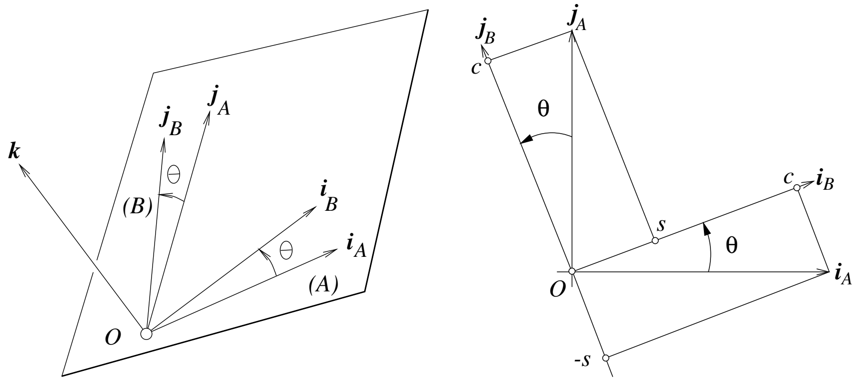


Figure 5.5. Two coordinate frames separated by a rotation of angle θ about their common $\boldsymbol{k}$ basis vector. As shown in the right of the figure, $\boldsymbol{i}_A = c\boldsymbol{i}_B - s\boldsymbol{j}_B$ and $\boldsymbol{i}_A = s\boldsymbol{i}_B + c\boldsymbol{j}_B$, where $c = \cos\theta$ and $s = \sin\theta$.

it can be shown that any rotation matrix can be written as the product of three elementary rotations about the $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ vectors of some coordinate system.

Let us go back to characterizing the change of coordinates associated with an arbitrary rotation matrix. Writing

$$\overrightarrow{OP} = (\,\boldsymbol{i}_A \quad \boldsymbol{j}_A \quad \boldsymbol{k}_A\,) \begin{pmatrix} {}^A x \\ {}^A y \\ {}^A z \end{pmatrix} = (\,\boldsymbol{i}_B \quad \boldsymbol{j}_B \quad \boldsymbol{k}_B\,) \begin{pmatrix} {}^B x \\ {}^B y \\ {}^B z \end{pmatrix}$$

in the frame (B) yields immediately

$${}^B P = {}^B_A\mathcal{R}\,{}^A P$$

since the rotation matrix ${}^B_B\mathcal{R}$ is obviously the identity. Note how the subscript matches the following superscript. This property remains true for more general coordinate changes and it can be used after some practice to reconstruct the corresponding formulas without calculations.

It is easy to show (see exercises) that rotation matrices are characterized by the following properties: (1) the inverse of a rotation matrix is equal to its transpose, and (2) its determinant is equal to 1. By definition, the columns of a rotation matrix form a right-handed orthonormal coordinate system. It follows from property (1) that their rows also form such a coordinate system.

It should be noted that the set of rotation matrices, equipped with the matrix product, forms a *group*, i.e., the product of two rotation matrices is also a rotation matrix (this is intuitively obvious and easily verified analytically); the matrix product is associative; there is a unit element, the 3×3 identity matrix Id; and every rotation matrix $\mathcal{R}$ admits an inverse $\mathcal{R}^{-1} = \mathcal{R}^T$ such that $\mathcal{R}\mathcal{R}^{-1} = \mathcal{R}^{-1}\mathcal{R} = \text{Id}$.

When the origins and the basis vectors of the two coordinate systems are different, we say that that the frames are separated by a general *rigid transformation* (Figure 5.6), and we have

$${}^BP = {}^B_A\mathcal{R}\,{}^AP + {}^BO_A. \tag{5.1.4}$$

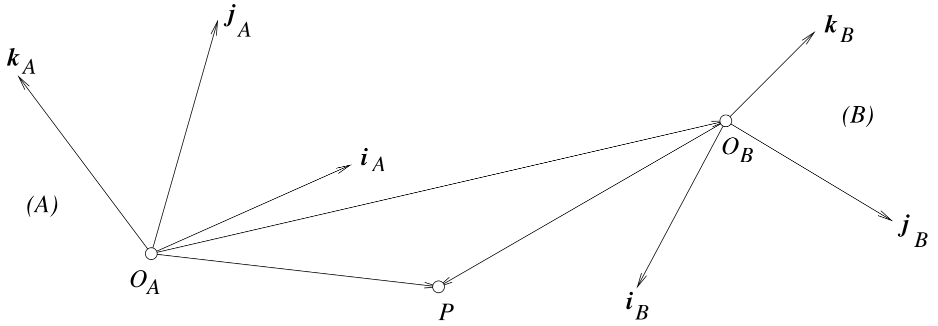


Figure 5.6. Coordinate changes between two frames: general rigid transformation.

Homogeneous coordinates can be used to rewrite (5.1.4) as a matrix product: let us first note that matrices can be multiplied in blocks, i.e., if

$$\mathcal{A} = \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ \mathcal{A}_{21} & \mathcal{A}_{22} \end{pmatrix} \quad \text{and} \quad \mathcal{B} = \begin{pmatrix} \mathcal{B}_{11} & \mathcal{B}_{12} \\ \mathcal{B}_{21} & \mathcal{B}_{22} \end{pmatrix}, \tag{5.1.5}$$

where the number of columns of the sub-matrices $\mathcal{A}_{11}$ and $\mathcal{A}_{21}$ (resp. $\mathcal{A}_{12}$ and $\mathcal{A}_{22}$) is equal to the number of rows of $\mathcal{B}_{11}$ and $\mathcal{B}_{12}$ (resp. $\mathcal{B}_{21}$ and $\mathcal{B}_{22}$), then

$$\mathcal{A}\mathcal{B} = \begin{pmatrix} \mathcal{A}_{11}\mathcal{B}_{11} + \mathcal{A}_{12}\mathcal{B}_{21} & \mathcal{A}_{11}\mathcal{B}_{12} + \mathcal{A}_{12}\mathcal{B}_{22} \\ \mathcal{A}_{21}\mathcal{B}_{11} + \mathcal{A}_{22}\mathcal{B}_{21} & \mathcal{A}_{21}\mathcal{B}_{12} + \mathcal{A}_{22}\mathcal{B}_{22} \end{pmatrix}.$$

For example, we have

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{pmatrix} = \begin{pmatrix} r_{11}c_{11} + r_{12}c_{21} + r_{13}c_{31} & r_{11}c_{12} + r_{12}c_{22} + r_{13}c_{32} \\ r_{21}c_{11} + r_{22}c_{21} + r_{23}c_{31} & r_{21}c_{12} + r_{22}c_{22} + r_{23}c_{32} \\ r_{31}c_{11} + r_{32}c_{21} + r_{33}c_{31} & r_{31}c_{12} + r_{32}c_{22} + r_{33}c_{32} \end{pmatrix}$$

$$= \begin{array}{|ccc|}\hline r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ \hline r_{31} & r_{32} & r_{33} \\ \hline \end{array} \begin{array}{|c|c|}\hline c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \\ \hline \end{array} = \begin{pmatrix} \begin{pmatrix} r_{11} \; r_{12} \; r_{13} \\ r_{21} \; r_{22} \; r_{23} \end{pmatrix} \begin{pmatrix} c_{11} \\ c_{21} \\ c_{31} \end{pmatrix} & \begin{pmatrix} r_{11} \; r_{12} \; r_{13} \\ r_{21} \; r_{22} \; r_{23} \end{pmatrix} \begin{pmatrix} c_{12} \\ c_{22} \\ c_{32} \end{pmatrix} \\ \begin{pmatrix} r_{31} \; r_{32} \; r_{33} \end{pmatrix} \begin{pmatrix} c_{11} \\ c_{21} \\ c_{31} \end{pmatrix} & \begin{pmatrix} r_{31} \; r_{32} \; r_{33} \end{pmatrix} \begin{pmatrix} c_{12} \\ c_{22} \\ c_{32} \end{pmatrix} \end{pmatrix}.$$

In particular, (5.1.5) allows us to rewrite the change of coordinates (5.1.4) as

$$\begin{pmatrix} {}^BP \\ 1 \end{pmatrix} = {}^B_A\mathcal{T} \begin{pmatrix} {}^AP \\ 1 \end{pmatrix}, \quad \text{where} \quad {}^B_A\mathcal{T} \overset{\text{def}}{=} \begin{pmatrix} {}^B_A\mathcal{R} & {}^BO_A \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{5.1.6}$$

and $\mathbf{0} = (0, 0, 0)^T$. In other words, using homogeneous coordinates allows us to write a general change of coordinates as the product of a 4×4 matrix and a 4-vector. It is easy to show that the set of rigid transformations defined by (5.1.6), equipped with the matrix product operation is also a group (see exercises).

A rigid transformation maps a coordinate system onto another one. In a given coordinate frame (F), a rigid displacement can also be considered as a mapping between points, i.e., a point P is mapped onto the point P' such that

$$ {}^F P' = \mathcal{R}\,{}^F P + \boldsymbol{t} \iff \begin{pmatrix} {}^F P' \\ 1 \end{pmatrix} = \begin{pmatrix} \mathcal{R} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} {}^F P \\ 1 \end{pmatrix}, \qquad (5.1.7) $$

where $\mathcal{R}$ is a rotation matrix and $\boldsymbol{t}$ is an element of $\mathbb{R}^3$ (Figure 5.7). The set of rigid transformations considered as mappings of $\mathbb{E}^3$ onto itself and equipped with the law of composition is once again easily shown to form a group. It is also easy to show that rigid transformations preserve the distance between points and the angle between vectors. On the other hand, the 4×4 matrix associated with a rigid transformation depends on the choice of (F) (see exercises).

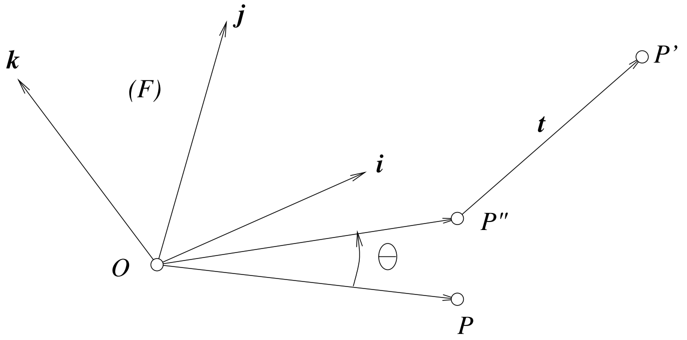


Figure 5.7. A rigid transformation maps the point P onto the point P'' through a rotation $\mathcal{R}$ before mapping P'' onto P' via a translation $\boldsymbol{t}$. In the example shown in this figure, $\mathcal{R}$ is a rotation of angle θ about the $\boldsymbol{k}$ axis of the coordinate system (F).

For example, let us consider the rotation of angle θ about the $\boldsymbol{k}$ axis of the frame (F). As shown in the exercises, this mapping can be represented by

$$ {}^F P' = \mathcal{R}\,{}^F P, \quad \text{where} \quad \mathcal{R} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}. $$

In particular, if (F') is the coordinate system obtained by applying this rotation to (F), we have, according to (5.1.3), ${}^{F'} P = {}^{F'}_{F}\mathcal{R}\,{}^F P$, and $\mathcal{R} = {}^{F'}_{F}\mathcal{R}^{-1}$. More generally, the matrix representing the change of coordinates between two frames is the inverse of the matrix mapping the first frame onto the second one (see exercises).

What happens when $\mathcal{R}$ is replaced by an arbitrary 3×3 matrix $\mathcal{A}$? Equation (5.1.7) still represents a mapping between points (or a change of coordinates between frames), but this time lengths and angles may not be preserved anymore (equivalently, the new coordinate system does not necessarily have orthogonal axes with unit length). We say that the 4×4 matrix

$$\mathcal{T} = \begin{pmatrix} \mathcal{A} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

represents an *affine transformation*. When $\mathcal{T}$ is allowed to be completely arbitrary, we say that we have a *projective transformation*. Affine and projective transformations also form groups, and they will be given a more thorough treatment later in the book.

5.2 Geometric Camera Parameters

We saw in Chapter **??** that the coordinates (x, y, z) of a scene point P observed by a pinhole camera are related to its image coordinates (x', y') by the perspective equation (**??**). In reality, this equation is only valid when all distances are measured in the camera's reference frame, and image coordinates have their origin at the principal point where the axis of symmetry of the camera pierces its retina. In practice, the world and camera coordinate systems are related by a set of physical parameters, such as the focal length of the lens, the size of the pixels, the position of the principal point, and the position and orientation of the camera.

This section identifies these parameters. We will distinguish the *intrinsic* parameters, that relate the camera's coordinate system to the idealized coordinate system used in Chapter **??**, from the *extrinsic* parameters, that relate the camera's coordinate system to a fixed world coordinate system and specify its position and orientation in space.

Before proceeding, let us note that we will ignore in the rest of this chapter the fact that for cameras equipped with a lens, a point will only be in focus when its depth and the distance betweem the optical center of the camera and its image plane obey the thin lens equation (**??**). Likewise, the non-linear aberrations associated with real lenses are not taken into account by (**??**). We will neglect these aberrations in most of the chapter but will consider radial distortion in Section 5.3.2.

5.2.1 Intrinsic Parameters

We can associate with a camera two different image planes: the first one is a normalized plane located at a unit distance from the pinhole. We attach to this plane its own coordinate system with an origin located at the point $\hat{C}$ where the optical axis pierces it (Figure 5.8). The perspective projection equation (**??**) can be written in this normalized coordinate system as

$$\begin{cases} \hat{u} = \dfrac{x}{z} \\ \hat{v} = \dfrac{y}{z} \end{cases} \Longleftrightarrow \hat{\boldsymbol{p}} = \frac{1}{z} \begin{pmatrix} \mathrm{Id} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{P} \\ 1 \end{pmatrix}, \tag{5.2.1}$$

where $\hat{\boldsymbol{p}} \overset{\text{def}}{=} (\hat{u}, \hat{v}, 1)^T$ is the vector of homogeneous coordinates of the projection $\hat{p}$ of the point P into the normalized image plane.

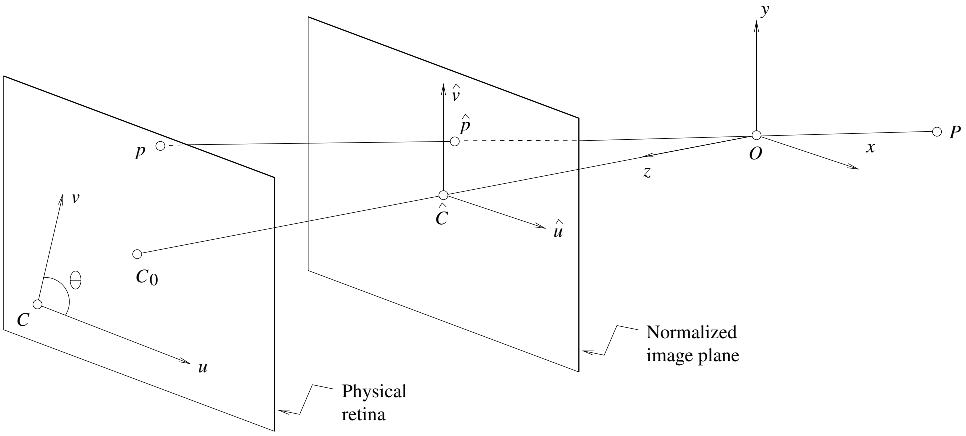


Figure 5.8. Physical and normalized image coordinate systems.

The physical retina of the camera is in general different (Figure 5.8): it is located at a distance $f \neq 1$ from the pinhole,[3] and the image coordinates (u, v) of the image point p are usually expressed in pixel units (instead of, say, meters). In addition, pixels are normally rectangular instead of square, so the camera has two additional scale parameters k and l, and

$$\begin{cases} u = kf\dfrac{x}{z}, \\ v = lf\dfrac{y}{z}. \end{cases} \tag{5.2.2}$$

Let us talk units for a second: f is a distance, expressed in meters for example, and a pixel will have dimensions $\frac{1}{k} \times \frac{1}{l}$, where k and l are expressed in $\text{pixel} \times m^{-1}$. The parameters k, l and f are not independent, and they can be replaced by the magnifications $\alpha = kf$ and $\beta = lf$ expressed in pixel units.

Now, in general, the actual origin of the camera coordinate system is at a corner C of the retina (e.g., in the case depicted in Figure 5.8, the lower-left corner, or sometimes the upper-left corner, when the image coordinates are the row and column indices of a pixel) and not at its center, and the center of the CCD matrix usually does not coincide with the principal point C_0. This adds two parameters u_0 and v_0 that define the position (in pixel units) of C_0 in the retinal coordinate system. Thus, (5.2.2) is replaced by

---

[3] From now on we will assume that the camera is focused at infinity so the distance between the pinhole and the image plane is equal to the focal length.

$$\begin{cases} u = \alpha\dfrac{x}{z} + u_0, \\ v = \beta\dfrac{y}{z} + v_0. \end{cases} \tag{5.2.3}$$

Finally, the camera coordinate system may also be skewed, due to some manufacturing error, so the angle θ between the two image axes is not equal to (but of course not very different from either) 90 degrees. In this case, it is easy to show (see exercises) that (5.2.3) transforms into

$$\begin{cases} u = \alpha\dfrac{x}{z} - \alpha\cot\theta\dfrac{y}{z} + u_0, \\ v = \dfrac{\beta}{\sin\theta}\dfrac{y}{z} + v_0. \end{cases} \tag{5.2.4}$$

Combining (5.2.1) and (5.2.4) now allows us to write the change in coordinates between the physical image frame and the normalized one as a planar affine transformation:

$$\boldsymbol{p} = \mathcal{K}\hat{\boldsymbol{p}}, \quad \text{where} \quad \boldsymbol{p} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad \text{and} \quad \mathcal{K} \overset{\text{def}}{=} \begin{pmatrix} \alpha & -\alpha\cot\theta & u_0 \\ 0 & \dfrac{\beta}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Putting it all together, we obtain

$$\boldsymbol{p} = \frac{1}{z}\mathcal{M}\boldsymbol{P}, \quad \text{where} \quad \mathcal{M} \overset{\text{def}}{=} (\mathcal{K} \quad \boldsymbol{0}) \tag{5.2.5}$$

and $\boldsymbol{P}$ denotes this time the *homogeneous* coordinate vector of P in the camera coordinate system: homogeneous coordinates have allowed us to represent the perspective projection mapping by the 3×4 matrix $\mathcal{M}$.

Note that the physical size of the pixels and the skew are always fixed for a given camera and frame grabber, and they can in principle be measured during manufacturing (this information may of course not be available, in the case of stock film footage for example, or when the frame grabber's digitization rate is unknown and different from 1). For zoom lenses, the focal length and possibly the optical center may vary with time. As mentioned earlier, simply changing the focus of the camera will also affect the magnification since it will change the lens-to-retina distance, but we will ignore this effect in the sequel.

5.2.2 Extrinsic Parameters

We consider in this section the case where the camera frame (C) is distinct from the world frame (W). Noting that

$$ {}^{C}P = \left({}^{C}_{W}\mathcal{R} \quad {}^{C}O_W \right) \begin{pmatrix} {}^{W}P \\ 1 \end{pmatrix} $$

and substituting in (5.2.5) yields

$$ \boldsymbol{p} = \frac{1}{z}\mathcal{M}\boldsymbol{P}, \quad \text{where} \quad \mathcal{M} = \mathcal{K}\left(\mathcal{R} \quad \boldsymbol{t} \right), \tag{5.2.6} $$

$\mathcal{R} = {}^{C}_{W}\mathcal{R}$ is a rotation matrix, $\boldsymbol{t} = {}^{C}O_W$ is a translation vector, and $\boldsymbol{P}$ denotes the vector of homogeneous coordinates of P in the frame (W).

We will often write the general perspective projection equation as $z\boldsymbol{p} = \mathcal{M}\boldsymbol{P}$, or even, slightly abusing the notation, as $\boldsymbol{p} = \mathcal{M}\boldsymbol{P}$, with the convention that a vector of homogeneous coordinates is only defined up to scale, and the actual image coordinates of the image point p being defined as u/w and v/w if $\boldsymbol{p} = (u, v, w)^T$. In this setting, the matrix $\mathcal{M}$ is also defined up to scale, with 11 free coefficients. Note that there are 5 intrinsic parameters (α, β, u_0, v_0 and θ) and 6 extrinsic parameters (the three angles defining $\mathcal{R}$ and the three coordinates of $\boldsymbol{t}$), which matches the number of independent coefficients of $\mathcal{M}$.

The matrix $\mathcal{M}$ can of course be rewritten explicitly as a function of the intrinsic and extrinsic parameters of the camera, namely

$$ \mathcal{M} = \begin{pmatrix} \alpha\boldsymbol{r}_1^T - \alpha\cot\theta\boldsymbol{r}_2^T + u_0\boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + u_0 t_z \\ \dfrac{\beta}{\sin\theta}\boldsymbol{r}_2^T + v_0\boldsymbol{r}_3^T & \dfrac{\beta}{\sin\theta}t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix}, \tag{5.2.7} $$

where $\boldsymbol{r}_1^T$, $\boldsymbol{r}_2^T$ and $\boldsymbol{r}_3^T$ denote the three rows of the matrix $\mathcal{R}$ and t_x, t_y and t_z are the coordinates of the vector $\boldsymbol{t}$ in the frame attached to the camera. If $\mathcal{R}$ is written as the product of three elementary rotations, the vectors $\boldsymbol{r}_i$ $(i = 1, 2, 3)$ can of course be written explicitly in terms of the corresponding three angles.

It is worth noting that the matrix $\mathcal{M}$ determines the coordinate vector $\boldsymbol{C}$ of the camera's optical center in the world coordinate system. Indeed, as shown in the exercises, $\boldsymbol{C}$ verifies

$$ \mathcal{M}\begin{pmatrix} \boldsymbol{C} \\ 1 \end{pmatrix} = 0. $$

(Intuitively this is rather obvious since the optical center is the only point whose image is not uniquely defined.) In particular, if $\mathcal{M} = \left(\mathcal{A} \quad \boldsymbol{b} \right)$ then $\boldsymbol{C} = -\mathcal{A}^{-1}\boldsymbol{b}$.

5.2.3 A Characterization of Perspective Projection Matrices

We say that a 3×4 matrix that can be written (up to scale) as (5.2.6) or equivalently (5.2.7) for some set of intrinsic and extrinsic parameters is a *perspective projection* *matrix.* It is of practical interest to put some restrictions on the intrinsic parameters of a camera since, as noted earlier, some of these parameters will be fixed and may be known. In particular, we will say that a 3×4 matrix is a *zero-skew perspective projection matrix* when it can be rewritten (up to scale) as (5.2.7) with $\theta = \pi/2$, and that it is a *perspective projection matrix with zero skew and unit aspect-ratio* when it can be rewritten (up to scale) as (5.2.7) with $\theta = \pi/2$ and $\alpha = \beta$. Of course, a camera with *known* non-zero skew and non-unit aspect-ratio can be transformed into a camera with zero skew and unit aspect-ratio by an appropriate change of image coordinates. Are arbitrary 3×4 matrices perspective projection matrices? The following theorem answers this question.

Theorem 2: *Let* $\mathcal{M} = (\mathcal{A} \quad \boldsymbol{b})$ *be a* 3×4 *matrix and let* $\boldsymbol{a}_i^T$ $(i = 1, 2, 3)$ *denote the rows of the matrix* $\mathcal{A}$ *formed by the three leftmost columns of* $\mathcal{M}$*.*

- *A necessary and sufficient condition for* $\mathcal{M}$ *to be a perspective projection matrix is that* $\mathrm{Det}(\mathcal{A}) \neq 0$*.*
- *A necessary and sufficient condition for* $\mathcal{M}$ *to be a zero-skew perspective projection matrix is that* $\mathrm{Det}(\mathcal{A}) \neq 0$ *and*

$$(\boldsymbol{a}_1 \times \boldsymbol{a}_3) \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3) = 0.$$

- *A necessary and sufficient condition for* $\mathcal{M}$ *to be a perspective projection matrix with zero skew and unit aspect-ratio is that* $\mathrm{Det}(\mathcal{A}) \neq 0$ *and*

$$\begin{cases} (\boldsymbol{a}_1 \times \boldsymbol{a}_3) \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3) = 0, \\ (\boldsymbol{a}_1 \times \boldsymbol{a}_3) \cdot (\boldsymbol{a}_1 \times \boldsymbol{a}_3) = (\boldsymbol{a}_2 \times \boldsymbol{a}_3) \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3). \end{cases}$$

The conditions of the theorem are clearly necessary: according to (5.2.6), we have $\mathcal{A} = \mathcal{K}\mathcal{R}$, thus the determinants of $\mathcal{A}$ and $\mathcal{K}$ are the same and $\mathcal{A}$ is non-singular. Further, a simple calculation shows that the rows of $\mathcal{K}\mathcal{R}$ in (5.2.7) satisfy the conditions of the theorem under the various assumptions imposed by its statement. Proofs that they are also sufficient can be found in [**?**; **?**] and in the exercises. Note that when the conditions of the theorem are satisfied, there are exactly four sets of intrinsic and extrinsic parameters satisfying (5.2.7), see [**?**; **?**] and Section 5.3.1.

5.3 Calibration Methods

This section introduces various techniques for estimating the intrinsic and extrinsic parameters of a camera, a process known as geometric camera calibration. Specifically, suppose that a camera observes n geometric features such as points or lines with known positions in some fixed world coordinate system. This section addresses the problem of (1) computing the perspective projection matrix $\mathcal{M}$ associated with the camera in this coordinate system, then (2) computing the intrinsic and extrinsic parameters of the camera from this matrix. Once a camera has been calibrated, it is possible to associate with any image point a well-defined ray passing through this point and the camera's optical center, and to conduct quantitative three-dimensional measurements from digitized pictures [**?**].

5.3.1 A Linear Approach to Camera Calibration

Let us first assume that our camera has non-zero skew. According to Theorem 2, the matrix $\mathcal{M}$ is not singular but otherwise arbitrary. If the 4-vectors $\boldsymbol{P}_i$ $(i = 1, \ldots, n)$ and $\boldsymbol{m}_j^T$ $(j = 1, 2, 3)$ denote respectively the homogeneous coordinate vectors of the points P_i and the rows of the matrix $\mathcal{M}$, we can express the position of the image of each point as

$$\left\{\begin{array}{l} u_i = \dfrac{\boldsymbol{m}_1 \cdot \boldsymbol{P}_i}{\boldsymbol{m}_3 \cdot \boldsymbol{P}_i}, \\[2ex] v_i = \dfrac{\boldsymbol{m}_2 \cdot \boldsymbol{P}_i}{\boldsymbol{m}_3 \cdot \boldsymbol{P}_i}, \end{array}\right. \Longleftrightarrow \left\{\begin{array}{l} (\boldsymbol{m}_1 - u_i \boldsymbol{m}_3) \cdot \boldsymbol{P}_i = 0, \\ (\boldsymbol{m}_2 - v_i \boldsymbol{m}_3) \cdot \boldsymbol{P}_i = 0. \end{array}\right.$$

Collecting these constraints for all points yields a system of $2n$ homogeneous linear equations in the twelve coefficients of the matrix $\mathcal{M}$, namely,

$$\mathcal{P}\boldsymbol{m} = 0, \quad \text{where} \quad \mathcal{P} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{P}_1^T & \boldsymbol{0}^T & -u_1 \boldsymbol{P}_1^T \\ \boldsymbol{0}^T & \boldsymbol{P}_1^T & -v_1 \boldsymbol{P}_1^T \\ \ldots & \ldots & \ldots \\ \boldsymbol{P}_n^T & \boldsymbol{0}^T & -u_n \boldsymbol{P}_n^T \\ \boldsymbol{0}^T & \boldsymbol{P}_n^T & -v_n \boldsymbol{P}_n^T \end{pmatrix} \quad \text{and} \quad \boldsymbol{m} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \\ \boldsymbol{m}_3 \end{pmatrix} = 0. \tag{5.3.1}$$

When $n \geq 6$, the system of equations (5.3.1) is in general *overconstrained*, i.e., there is no non-zero vector $\boldsymbol{m} \in \mathbb{R}^{12}$ that satisfies exactly these equations. On the other hand, the zero vector is always a solution. The *linear least-squares* literature, as briefly discussed in the insert next page, provides methods for computing the value of the *unit* vector $\boldsymbol{m}$ that minimizes $|\mathcal{P}\boldsymbol{m}|^2$. In particular, estimating the vector $\boldsymbol{m}$ (hence the matrix $\mathcal{M}$) reduces to computing the eigenvectors and eigenvalues of the 12×12 matrix $\mathcal{P}^T\mathcal{P}$.

Technique: Linear Least Squares Methods

Let us consider a system of n linear equations in p unknowns:

$$\left\{\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1p}x_p = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2p}x_p = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{np}x_p = b_n \end{array}\right. \Leftrightarrow \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1p} \\ a_{21} & a_{22} & \ldots & a_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{np} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_p \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}. \tag{5.3.2}$$

Let $\mathcal{A}$ denote the $n \times p$ matrix with coefficients a_{ij}, and let $\boldsymbol{x} = (x_1, \ldots, x_p)^T$ and $\boldsymbol{b} = (b_1, \ldots, b_n)^T$. We know from linear algebra that (in general):

1. when $n < p$, there exists an $(p-n)$-dimensional vector space of vectors $\boldsymbol{x}$ that are solutions of (5.3.2);
2. when $n = p$, there is a unique solution;
3. when $n > p$, there is no solution.

This statement is true when the rank of $\mathcal{A}$ is maximal, i.e., equal to $\min(n,p)$ (this is what we mean by "in general"). When the rank is lower, there exists a higher-dimensional set of solutions.

Here we will consider the overconstrained case $n > p$. Since there is no exact solution in this case, we will content ourselves with finding the vector $\boldsymbol{x}$ that minimizes the error measure

$$E \stackrel{\text{def}}{=} \sum_{i=1}^{n} (a_{i1}x_1 + \ldots + a_{ip}x_p - b_i)^2 = |\mathcal{A}\boldsymbol{x} - \boldsymbol{b}|^2.$$

E is proportional to the mean-squared error associated with the equations, hence the name of least-squares methods given to techniques for minimizing E.

Now, we can write $E = |\boldsymbol{e}^T\boldsymbol{e}|$, where $\boldsymbol{e} \stackrel{\text{def}}{=} \mathcal{A}\boldsymbol{x} - \boldsymbol{b}$. To find the vector $\boldsymbol{x}$ minimizing E, we write that the derivatives of this error measure with respect to the coordinates x_i $(i = 1, \ldots, p)$ of $\boldsymbol{x}$ must be zero, i.e.,

$$\frac{\partial E}{\partial x_i} = 2\frac{\partial \boldsymbol{e}}{\partial x_i} \cdot \boldsymbol{e} = 0 \quad \text{for} \quad i = 1, \ldots, p.$$

But if the vectors $\boldsymbol{c}_i$ $(i = 1, \ldots, p)$ denote the columns of $\mathcal{A}$, we have

$$\frac{\partial \boldsymbol{e}}{\partial x_i} = \frac{\partial}{\partial x_i}\left[\begin{pmatrix} \boldsymbol{c}_1 & \ldots & \boldsymbol{c}_p \end{pmatrix}\begin{pmatrix} x_1 \\ \ldots \\ x_p \end{pmatrix} - \boldsymbol{b}\right] = \frac{\partial}{\partial x_i}(x_1\boldsymbol{c}_1 + \ldots + x_p\boldsymbol{c}_p - \boldsymbol{b}) = \boldsymbol{c}_i.$$

In particular, the constraint $\partial E/\partial x_i = 0$ implies that $\boldsymbol{c}_i^T(\mathcal{A}\boldsymbol{x} - \boldsymbol{b}) = 0$, and stacking the constraints associated with the p coordinates of $\boldsymbol{x}$ yields

$$\mathbf{0} = \begin{pmatrix} \boldsymbol{c}_1^T \\ \ldots \\ \boldsymbol{c}_p^T \end{pmatrix}(\mathcal{A}\boldsymbol{x} - \boldsymbol{b}) = \mathcal{A}^T(\mathcal{A}\boldsymbol{x} - \boldsymbol{b}) \Longleftrightarrow \mathcal{A}^T\mathcal{A}\boldsymbol{x} = \mathcal{A}^T\boldsymbol{b}.$$

The equations in this linear system are called the normal equations. When $\mathcal{A}$ has maximal rank p, the matrix $\mathcal{A}^T\mathcal{A}$ is easily shown to be invertible, and the solution of the least-squares problem can be written as

$$\boldsymbol{x} = \mathcal{A}^{\dagger}\boldsymbol{b} \quad \text{where} \quad \mathcal{A}^{\dagger} \stackrel{\text{def}}{=} [(\mathcal{A}^T\mathcal{A})^{-1}\mathcal{A}^T].$$

The $p \times p$ matrix $\mathcal{A}^\dagger$ is called the pseudoinverse of $\mathcal{A}$. It coincides with $\mathcal{A}^{-1}$ when the matrix $\mathcal{A}$ is square and non-singular. Linear least-squares problems can be solved without explicitly computing the pseudoinverse, using for example QR decomposition or singular value decomposition techniques, which are known to be better behaved numerically.

Let us now consider a slightly different problem, where we have a system of n *homogeneous* linear equations in p unknowns:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1p}x_p = 0 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2p}x_p = 0 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{np}x_p = 0 \end{cases} \Leftrightarrow \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1p} \\ a_{21} & a_{22} & \ldots & a_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{np} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_p \end{pmatrix} = \mathbf{0}. \quad (5.3.3)$$

As before, we denote by $\mathcal{A}$ the $n \times p$ matrix with coefficients a_{ij}, and define $\boldsymbol{x} = (x_1, \ldots, x_p)^T$. When $n = p$ and the matrix $\mathcal{A}$ is non-singular, the system (5.3.3) admits as a unique solution $\boldsymbol{x} = \mathbf{0}$. Conversely, when $n \geq p$, non-trivial (i.e., non-zero) solutions may only exist when $\mathcal{A}$ is singular.

In this context, minimizing the error measure

$$E \stackrel{\text{def}}{=} |\mathcal{A}\boldsymbol{x}|^2 = \sum_{i=1}^{n} [\boldsymbol{a}_i \cdot \boldsymbol{x}]^2$$

only makes sense when some constraint is imposed on the solution $\boldsymbol{x}$ since $\boldsymbol{x} = \mathbf{0}$ yields the zero global minimum of E.

Since, by homogeneity, $E(\lambda\boldsymbol{x}) = \lambda^2 E(\boldsymbol{x})$, it is reasonable to minimize E under the constraint $|\boldsymbol{x}|^2 = 1$, which avoids the trivial solution and forces the uniqueness of the result.

Let us have another look at the error $E = \boldsymbol{x}^T(\mathcal{A}^T\mathcal{A})\boldsymbol{x}$. The $p \times p$ matrix $\mathcal{A}^T\mathcal{A}$ is symmetric positive semidefinite, and it can be diagonalized in an orthonormal basis of eigenvectors $\boldsymbol{e}_i$ $(i = 1, \ldots, p)$ associated with the eigenvalues $0 \leq \lambda_1 \leq \ldots \leq \lambda_p$. Now we can write any unit vector $\boldsymbol{x}$ as $\boldsymbol{x} = \mu_1\boldsymbol{e}_1 + \ldots + \mu_p\boldsymbol{e}_p$ for some μ_i $(i = 1, \ldots, p)$ such that $\mu_1^2 + \ldots + \mu_p^2 = 1$. We have

$$E(\boldsymbol{x}) - E(\boldsymbol{e}_1) = \boldsymbol{x}^T(\mathcal{A}^T\mathcal{A})\boldsymbol{x} - \boldsymbol{e}_1^T(\mathcal{A}^T\mathcal{A})\boldsymbol{e}_1 = \lambda_1\mu_1^2 + \ldots + \lambda_p\mu_p^2 - \lambda_1 \geq \lambda_1(\mu_1^2 + \ldots + \mu_p^2 - 1) = 0.$$

It follows that the unit vector $\boldsymbol{x}$ minimizing the least-squares error E is the eigenvector $\boldsymbol{e}_1$ associated with the minimum eigenvalue of $\mathcal{A}^T\mathcal{A}$ and the corresponding minimum value of E is λ_1.

Various methods are available for computing the eigenvectors and eigenvalues of a symmetric matrix, including Jacobi transformations and reduction to tridiagonal form followed by QR decomposition.

It should finally be noted that least-squares minimization admits a statistical interpretation in terms of maximum likelihood when the coordinates of the data points are modelled as random variables obeying a normal distribution. We will come back to this interpretation in a latter chapter.

In the noise-free case, there will be a unique solution for the matrix $\mathcal{M}$ as long at the rank of the matrix $\mathcal{P}$ is equal to its maximum value of 11 (the matrix $\mathcal{P}$ is singular since by construction $\mathcal{P}\boldsymbol{m} = 0$). A degenerate point configuration will correspond to the case where the matrix has rank 10 or less, or equivalently the nullspace of the matrix has dimension two or greater. Let us consider a vector $\boldsymbol{l}$ in the nullspace and introduce the vectors formed by successive quadruples of its coordinates, i.e., $\boldsymbol{\lambda} = (l_1, l_2, l_3, l_4)^T$, $\boldsymbol{\mu} = (l_5, l_6, l_7, l_8)^T$ and $\boldsymbol{\nu} = (l_9, l_{10}, l_{11}, l_{12})^T$. Since $\boldsymbol{l}$ belongs to the nullspace we have

$$\boldsymbol{0} = \mathcal{P}\boldsymbol{l} = \begin{pmatrix} \boldsymbol{P}_1^T & \boldsymbol{0}^T & -u_1\boldsymbol{P}_1^T \\ \boldsymbol{0}^T & \boldsymbol{P}_1^T & -v_1\boldsymbol{P}_1^T \\ \dots & \dots & \dots \\ \boldsymbol{P}_n^T & \boldsymbol{0}^T & -u_n\boldsymbol{P}_n^T \\ \boldsymbol{0}^T & \boldsymbol{P}_n^T & -v_n\boldsymbol{P}_n^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\mu} \\ \boldsymbol{\nu} \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_1^T\boldsymbol{\lambda} - u_1\boldsymbol{P}_1^T\boldsymbol{\nu} \\ \boldsymbol{P}_1^T\boldsymbol{\mu} - v_1\boldsymbol{P}_1^T\boldsymbol{\nu} \\ \dots \\ \boldsymbol{P}_n^T\boldsymbol{\lambda} - u_n\boldsymbol{P}_n^T\boldsymbol{\nu} \\ \boldsymbol{P}_n^T\boldsymbol{\mu} - v_n\boldsymbol{P}_n^T\boldsymbol{\nu} \end{pmatrix},$$

or, equivalently, taking into account the values of u_i and v_i yields

$$\begin{cases} \boldsymbol{P}_i^T\boldsymbol{\lambda} - \dfrac{\boldsymbol{m}_1^T\boldsymbol{P}_i}{\boldsymbol{m}_3^T\boldsymbol{P}_i}\boldsymbol{P}_i^T\boldsymbol{\nu} = 0, \\ \boldsymbol{P}_i^T\boldsymbol{\mu} - \dfrac{\boldsymbol{m}_2^T\boldsymbol{P}_i}{\boldsymbol{m}_3^T\boldsymbol{P}_i}\boldsymbol{P}_i^T\boldsymbol{\nu} = 0, \end{cases} \quad \text{for} \quad i = 1, \dots, n.$$

We finally obtain after clearing the denominators and rearranging the terms:

$$\begin{cases} \boldsymbol{P}_i^T(\boldsymbol{m}_3\boldsymbol{\lambda}^T - \boldsymbol{m}_1\boldsymbol{\nu}^T)\boldsymbol{P}_i = 0, \\ \boldsymbol{P}_i^T(\boldsymbol{m}_3\boldsymbol{\mu}^T - \boldsymbol{m}_2\boldsymbol{\nu}^T)\boldsymbol{P}_i = 0, \end{cases} \quad \text{for} \quad i = 1, \dots, n. \tag{5.3.4}$$

As expected, the vector $\boldsymbol{l}$ associated with $\boldsymbol{\lambda} = \boldsymbol{m}_1$, $\boldsymbol{\mu} = \boldsymbol{m}_2$ and $\boldsymbol{\nu} = \boldsymbol{m}_3$ is a solution of these equations. Are there other solutions?

Let us first consider the case where the points P_i ($i = 1, \dots, n$) all lie in some plane Π, or equivalently, $\boldsymbol{\Pi} \cdot \boldsymbol{P}_i = 0$ for some 4-vector $\boldsymbol{\Pi}$. Clearly, choosing $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu})$ equal to $(\boldsymbol{\Pi}, \boldsymbol{0}, \boldsymbol{0})$, $(\boldsymbol{0}, \boldsymbol{\Pi}, \boldsymbol{0})$, or $(\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{\Pi})$, or any linear combination of these vectors will yield a solution of (5.3.4). In other words, the nullspace of $\mathcal{P}$ contains the four-dimensional vector space spanned by these vectors and $\boldsymbol{m}$. In practice, this means that coplanar points should not be used in calibration tasks.

In general, for a given non-zero value of the vector $\boldsymbol{l}$, the points P_i that satisfy (5.3.4) must lie on the curve where the two quadric surfaces defined by the corresponding equations intersect. A closer look at (5.3.4) reveals that the straight line where the planes defined by $\boldsymbol{m}_3 \cdot \boldsymbol{P} = 0$ and $\boldsymbol{\nu} \cdot \boldsymbol{P} = 0$ intersect lies on both quadrics. It can be shown that the intersection curve of these two surfaces consists of this line and of a twisted cubic curve Γ passing through the origin [**?**]. A twisted cubic is entirely determined by six points lying on it, and it follows that seven points chosen at random will not fall on Γ. Since, in addition, this curve passes through the origin, choosing $n \geq 6$ random points will in general guarantee that the matrix $\mathcal{P}$ has rank 11 and that the projection matrix can be recovered in a unique fashion.

Once the projection matrix $\mathcal{M}$ has been estimated, (5.2.7) can be used to recover the intrinsic and extrinsic parameters as follows. If we write as before $\mathcal{M} = (\,\mathcal{A} \quad \boldsymbol{b}\,)$, we have

$$\rho \begin{pmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \boldsymbol{a}_3^T \end{pmatrix} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T \\ \dfrac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T \\ \boldsymbol{r}_3^T \end{pmatrix}.$$

In particular, using the fact that the rows of a rotation matrix have unit length and are perpendicular to each other yields immediately

$$\begin{cases} \rho = \varepsilon / |\boldsymbol{a}_3|, \\ \boldsymbol{r}_3 = \rho \boldsymbol{a}_3, \\ u_0 = \rho^2 (\boldsymbol{a}_1 \cdot \boldsymbol{a}_3), \\ v_0 = \rho^2 (\boldsymbol{a}_2 \cdot \boldsymbol{a}_3), \end{cases} \tag{5.3.5}$$

where $\varepsilon = \mp 1$.

In addition, we have

$$\begin{cases} \rho^2 (\boldsymbol{a}_1 \times \boldsymbol{a}_3) = -\alpha \boldsymbol{r}_2 - \alpha \cot\theta \boldsymbol{r}_1, \\ \rho^2 (\boldsymbol{a}_2 \times \boldsymbol{a}_3) = \dfrac{\beta}{\sin\theta} \boldsymbol{r}_1, \end{cases} \quad \text{and} \quad \begin{cases} \rho^2 |\boldsymbol{a}_1 \times \boldsymbol{a}_3| = \dfrac{|\alpha|}{\sin\theta}, \\ \rho^2 |\boldsymbol{a}_2 \times \boldsymbol{a}_3| = \dfrac{|\beta|}{\sin\theta}, \end{cases} \tag{5.3.6}$$

since θ is always in the neighborhood of $\pi/2$ with a positive sine, and it follows that

$$\begin{cases} \cos\theta = -\varepsilon_u \varepsilon_v \dfrac{(\boldsymbol{a}_1 \times \boldsymbol{a}_3) \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)}{|\boldsymbol{a}_1 \times \boldsymbol{a}_3||\boldsymbol{a}_2 \times \boldsymbol{a}_3|}, \\ \alpha = \varepsilon_u \rho^2 |\boldsymbol{a}_1 \times \boldsymbol{a}_3| \sin\theta, \\ \beta = \varepsilon_v \rho^2 |\boldsymbol{a}_2 \times \boldsymbol{a}_3| \sin\theta, \end{cases} \tag{5.3.7}$$

where $\varepsilon_u = \alpha/|\alpha|$ and $\varepsilon_v = \beta/|\beta|$.

We can now compute $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ from the second equation in (5.3.6) as

$$\begin{cases} \boldsymbol{r}_1 = \dfrac{\rho^2 \sin\theta}{\beta} (\boldsymbol{a}_2 \times \boldsymbol{a}_3) = \dfrac{1}{|\boldsymbol{a}_2 \times \boldsymbol{a}_3|} (\boldsymbol{a}_2 \times \boldsymbol{a}_3), \\ \boldsymbol{r}_2 = \boldsymbol{r}_3 \times \boldsymbol{r}_1. \end{cases} \tag{5.3.8}$$

Note that there are four possible choices for the matrix $\mathcal{R}$ depending on the values of ε and ε_v.

Finally, the translation parameters are recovered by writing

$$\rho \begin{pmatrix} \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \dfrac{\beta}{\sin\theta} t_y + v_0 t_z \\ t_z \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

5.3.2 Taking Radial Distortion into Account

We have assumed so far that our camera was equipped with a perfect lens. As shown in Chapter **??**, real lenses suffer from a number of aberrations. In this section we follow Tsai [**?**] and show how to account for *radial distortion*, a type of aberration that depends on the distance between the imaged point and the optical axis and can be modelled as

$$\boldsymbol{p} = \begin{pmatrix} 1/\lambda & 0 & 0 \\ 0 & 1/\lambda & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathcal{M}\boldsymbol{P},$$

where λ is a polynomial function of $\hat{r}^2 \stackrel{\text{def}}{=} \hat{u}^2 + \hat{v}^2$, i.e., $\lambda = 1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4 + \ldots$.

Geometrically, radial distortion changes the distance between the image center and the image point $\boldsymbol{p}$ but it does not affect the direction of the vector joining these two points. This is called the *radial alignment constraint* by Tsai, and it can be expressed algebraically by writing

$$\lambda \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \dfrac{\boldsymbol{m}_1 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}} \\ \dfrac{\boldsymbol{m}_2 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}} \end{pmatrix} \Longrightarrow v(\boldsymbol{m}_1 \cdot \boldsymbol{P}) - u(\boldsymbol{m}_2 \cdot \boldsymbol{P}) = 0.$$

This is a linear constraint on the vectors $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$. Given n fiducial points we obtain n equations in the eight coefficients of the vectors $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, namely

$$\mathcal{Q}\boldsymbol{n} = 0, \quad \text{where} \quad \mathcal{Q} \stackrel{\text{def}}{=} \begin{pmatrix} v_1 \boldsymbol{P}_1^T & -u_1 \boldsymbol{P}_1^T \\ \ldots & \ldots \\ v_n \boldsymbol{P}_n^T & -u_n \boldsymbol{P}_n^T \end{pmatrix} \quad \text{and} \quad \boldsymbol{n} = \begin{pmatrix} \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \end{pmatrix}. \tag{5.3.9}$$

Note the similarity with the previous case. When $n \geq 8$, the system of equations (5.3.9) is in general overconstrained, and a solution with unit norm can be found using linear least squares.

We can as before determine the degenerate point configurations for which the vectors $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ will not be uniquely determined. The matrix $\mathcal{Q}$ has at most rank 7 since the vector $\boldsymbol{n}$ is in its nullspace. More generally, let us consider a vector $\boldsymbol{l}$ in the nullspace and the vectors $\boldsymbol{\lambda} = (l_1, l_2, l_3, l_4)^T$ and $\boldsymbol{\mu} = (l_5, l_6, l_7, l_8)^T$; we have

$$\boldsymbol{0} = \mathcal{Q}\boldsymbol{l} = \begin{pmatrix} v_1 \boldsymbol{P}_1^T & -u_1 \boldsymbol{P}_1^T \\ \ldots & \ldots \\ v_n \boldsymbol{P}_n^T & -u_n \boldsymbol{P}_n^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\mu} \end{pmatrix} = \begin{pmatrix} v_1 \boldsymbol{P}_1^T \boldsymbol{\lambda} - u_1 \boldsymbol{P}_1^T \boldsymbol{\mu} \\ \ldots \\ v_n \boldsymbol{P}_n^T \boldsymbol{\lambda} - u_n \boldsymbol{P}_n^T \boldsymbol{\mu} \end{pmatrix}.$$

Taking into account the values of u_i and v_i yields, after rearranging the terms and clearing the denominators,

$$\boldsymbol{P}_i^T (\boldsymbol{m}_2 \boldsymbol{\lambda}^T - \boldsymbol{m}_1 \boldsymbol{\mu}^T) \boldsymbol{P}_i = 0 \quad \text{for} \quad i = 1, \ldots, n. \tag{5.3.10}$$

The vector $\boldsymbol{l}$ associated with $\boldsymbol{\lambda} = \boldsymbol{m}_1$ and $\boldsymbol{\mu} = \boldsymbol{m}_2$ is of course a solution of these equations. When the points P_i ($i = 1, \ldots, n$) all lie in some plane Π, or equivalently,

$\boldsymbol{\Pi} \cdot \boldsymbol{P}_i = 0$ for some 4-vector $\boldsymbol{\Pi}$, we can choose $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ equal to $(\boldsymbol{\Pi}, \boldsymbol{0})$, $(\boldsymbol{0}, \boldsymbol{\Pi})$, or any linear combination of these two vectors, and construct a solution of (5.3.10). The nullspace of $\mathcal{P}$ contains the three-dimensional vector space spanned by these vectors and $\boldsymbol{l}$. Thus, as before, coplanar points should not be used in calibration.

More generally, for a given value of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the points P_i will form a degenerate configuration when they lie on the quadric surface defined by (5.3.10). Note that this surface contains the four straight lines defined by $\boldsymbol{\lambda} \cdot \boldsymbol{P} = \boldsymbol{\mu} \cdot \boldsymbol{P} = 0$, $\boldsymbol{\lambda} \cdot \boldsymbol{P} = \boldsymbol{m}_1 \cdot \boldsymbol{P} = 0$, $\boldsymbol{\mu} \cdot \boldsymbol{P} = \boldsymbol{m}_2 \cdot \boldsymbol{P} = 0$ and $\boldsymbol{m}_1 \cdot \boldsymbol{P} = \boldsymbol{m}_2 \cdot \boldsymbol{P} = 0$, and it must therefore be either two planes, a cone, a hyperboloid of one sheet or a hyperbolic paraboloid. In any case, for a large enough number of points in general position, there will be a unique solution to our least-squares problem.

Once $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ have been estimated, we can as before define the corresponding values of $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, b_1 and b_2 and we obtain the constraints

$$\rho \begin{pmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \end{pmatrix} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T \\ \dfrac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T \end{pmatrix}$$

In this setting, there are more unknowns (nine, i.e., the three coefficients of the rotation matrix, the two magnifications α and β, the coordinates u_0 and v_0 of the image center, the skew angle θ and the scale factor ρ) than equations (six scalar constraints corresponding to the two vector equations above).

As noted in [**?**], however, when the principal point is known we can take $u_0 = v_0 = 0$ and $\rho = 1$, and we obtain

$$\begin{cases} |\boldsymbol{a}_1| = \dfrac{|\alpha|}{\sin\theta}, \\ |\boldsymbol{a}_2| = \dfrac{|\beta|}{\sin\theta}. \end{cases}$$

In particular we have

$$\begin{cases} \cos\theta = -\varepsilon_u \varepsilon_v \dfrac{\boldsymbol{a}_1 \cdot \boldsymbol{a}_2}{|\boldsymbol{a}_1||\boldsymbol{a}_2|}, \\ \alpha = \varepsilon_u |\boldsymbol{a}_1| \sin\theta, \\ \beta = \varepsilon_v |\boldsymbol{a}_2| \sin\theta, \end{cases}$$

where as before $\varepsilon_u = \alpha/|\alpha|$ and $\varepsilon_v = \beta/|\beta|$. It is now easy to compute $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ and $\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2$, then, as before, the translation components.

Once the intrinsic and extrinsic parameters have been estimated, the radial distortion coefficients can themselves be recovered using linear least squares and the radial alignment constraint.

5.3.3 Using Straight Lines for Calibration

Points are not the only geometric image features that constrain the camera parameters. We characterize in this section the projection of straight lines into an image, and show how they can be used, in principle at least, to perform camera calibration. We must first recall some elementary notions of line geometry. Let us introduce the operator "$\wedge$" that associates with two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{R}^4$ their *exterior product* defined as the 6-vector

$$\boldsymbol{a} \wedge \boldsymbol{b} \stackrel{\text{def}}{=} \begin{pmatrix} a_1b_2 - a_2b_1 \\ a_1b_3 - a_3b_1 \\ a_1b_4 - a_4b_1 \\ a_2b_3 - a_3b_2 \\ a_2b_4 - a_4b_2 \\ a_3b_4 - a_4b_3 \end{pmatrix}.$$

Note the similarity with the cross-product operator that also associates with two vectors (3-vectors of course, instead of 4-vectors) $\boldsymbol{a}$ and $\boldsymbol{b}$ the vector formed by all the 2×2 minors of the matrix $(\boldsymbol{a}, \boldsymbol{b})$.

Let us assume a fixed coordinate system. Geometrically, the exterior product associates with the homogeneous coordinate vectors of two points A and B in $\mathbb{E}^3$ the vector $\boldsymbol{\Delta} = (\Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6)^T$ of *Plücker coordinates* of the line Δ joining them. To gain a better intuitive understanding of the situation, let us denote by O the origin of the coordinate system and by H its projection onto Δ (Figure 5.9), and let us identify the vectors $\overrightarrow{OA}$ and $\overrightarrow{OB}$ with their *non-homogeneous* coordinate vectors. It is easy to verify analytically (see exercises) that $\overrightarrow{AB} = -(\Delta_3, \Delta_5, \Delta_6)^T$ and $\overrightarrow{OA} \times \overrightarrow{OB} = \overrightarrow{OH} \times \overrightarrow{AB} = (\Delta_4, -\Delta_2, \Delta_1)^T$.

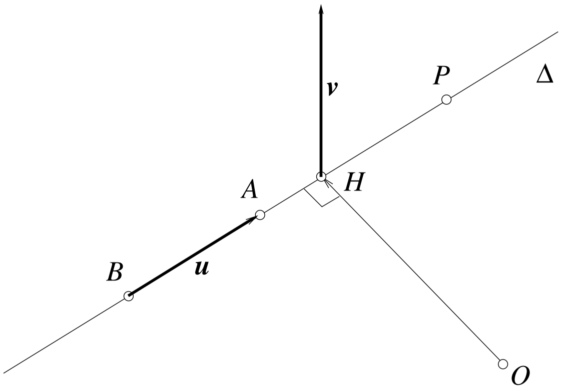


Figure 5.9. Geometric definition of line Plücker coordinates. In this figure $\boldsymbol{u} = (\Delta_3, \Delta_5, \Delta_6)^T$ and $\boldsymbol{v} = (\Delta_4, -\Delta_2, \Delta_1)^T$.

In turn, this implies that: (1) changing the position of A (or B) along Δ only changes the overall scale of $\boldsymbol{\Delta}$, so Plücker coordinates are homogeneous coordinates only defined up to scale but otherwise independent from the choice of the points A and B along Δ; and (2) the Plücker coordinates of a line obey the quadratic constraint

$$\Delta_1\Delta_6 - \Delta_2\Delta_5 + \Delta_3\Delta_4 = 0. \tag{5.3.11}$$

It is also possible to define an inner product on the set of all lines by the formula

$$(\boldsymbol{\Delta}|\boldsymbol{\Delta}') \stackrel{\text{def}}{=} \Delta_1\Delta'_6 + \Delta_6\Delta'_1 - \Delta_2\Delta'_5 - \Delta_5\Delta'_2 + \Delta_3\Delta'_4 + \Delta_4\Delta'_3.$$

Clearly, a 6-vector $\boldsymbol{\Delta}$ represents a line if and only if $(\boldsymbol{\Delta}|\boldsymbol{\Delta}) = 0$, and it can also be shown that a necessary and sufficient condition for two lines to be coplanar is that $(\boldsymbol{\Delta}|\boldsymbol{\Delta}') = 0$.

Let us now follow Faugeras and Papadopoulo [**?**] and show that the mapping between a line with Plücker coordinate vector $\boldsymbol{\Delta}$ and its image δ with homogeneous coordinates $\boldsymbol{\delta}$ can be represented by

$$\rho\boldsymbol{\delta} = \tilde{\mathcal{M}}\boldsymbol{\Delta}, \quad \text{where} \quad \tilde{\mathcal{M}} \stackrel{\text{def}}{=} \begin{pmatrix} (\boldsymbol{m}_2 \wedge \boldsymbol{m}_3)^T \\ (\boldsymbol{m}_3 \wedge \boldsymbol{m}_1)^T \\ (\boldsymbol{m}_1 \wedge \boldsymbol{m}_2)^T \end{pmatrix}, \tag{5.3.12}$$

$\boldsymbol{m}_1^T$, $\boldsymbol{m}_2^T$ and $\boldsymbol{m}_3^T$ denote as before the rows of $\mathcal{M}$ and ρ is an appropriate scale factor.

To prove this relation, let us consider a line Δ joining two points A and B, and denote by a and b the projections of these two points, with homogeneous coordinates $\boldsymbol{a} = \mathcal{M}\boldsymbol{a}$ and $\boldsymbol{b} = \mathcal{M}\boldsymbol{b}$. The points a and b lie on δ, thus $\boldsymbol{\delta} \cdot \boldsymbol{a} = \boldsymbol{\delta} \cdot \boldsymbol{b} = 0$. Hence, $\boldsymbol{\delta}$ (as an element of $\mathbb{R}^3$) is orthogonal to both $\mathcal{M}\boldsymbol{A}$ and $\mathcal{M}\boldsymbol{B}$ and must be parallel to their cross product. Thus we have

$$\rho\boldsymbol{\delta} = (\mathcal{M}\boldsymbol{A}) \times (\mathcal{M}\boldsymbol{B}) = \begin{pmatrix} (\boldsymbol{m}_2 \cdot \boldsymbol{A})(\boldsymbol{m}_3 \cdot \boldsymbol{B}) - (\boldsymbol{m}_3 \cdot \boldsymbol{A})(\boldsymbol{m}_2 \cdot \boldsymbol{B}) \\ (\boldsymbol{m}_3 \cdot \boldsymbol{A})(\boldsymbol{m}_1 \cdot \boldsymbol{B}) - (\boldsymbol{m}_1 \cdot \boldsymbol{A})(\boldsymbol{m}_3 \cdot \boldsymbol{B}) \\ (\boldsymbol{m}_1 \cdot \boldsymbol{A})(\boldsymbol{m}_2 \cdot \boldsymbol{B}) - (\boldsymbol{m}_2 \cdot \boldsymbol{A})(\boldsymbol{m}_1 \cdot \boldsymbol{B}) \end{pmatrix} \tag{5.3.13}$$

for some scale factor ρ.

Now, as shown in the exercises, the following identity holds for any 4-vectors $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$:

$$(\boldsymbol{a} \wedge \boldsymbol{b}) \cdot (\boldsymbol{c} \wedge \boldsymbol{d}) = (\boldsymbol{a} \cdot \boldsymbol{c})(\boldsymbol{b} \cdot \boldsymbol{d}) - (\boldsymbol{a} \cdot \boldsymbol{d})(\boldsymbol{b} \cdot \boldsymbol{c}).$$

Applying this identity to (5.3.13) yields

$$\rho\boldsymbol{\delta} = \begin{pmatrix} (\boldsymbol{m}_2 \wedge \boldsymbol{m}_3) \cdot (\boldsymbol{A} \wedge \boldsymbol{B}) \\ (\boldsymbol{m}_3 \wedge \boldsymbol{m}_1) \cdot (\boldsymbol{A} \wedge \boldsymbol{B}) \\ (\boldsymbol{m}_1 \wedge \boldsymbol{m}_2) \cdot (\boldsymbol{A} \wedge \boldsymbol{B}) \end{pmatrix},$$

and the result follows immediately.

The 3×6 matrix $\tilde{\mathcal{M}}$ is only defined up to scale with 17 coefficients. These parameters can be estimated as before via linear least squares (ignoring of course the non-linear constraints imposed by the fact that the rows of $\tilde{\mathcal{M}}$ are Plücker coordinates) when $n \geq 9$ (eliminating ρ in (5.3.12) yields two independent constraints per line).

Once $\tilde{\mathcal{M}}$ is known, we can recover $\mathcal{M}$ as well through linear least squares. Indeed, it is easily shown (see exercises) that

$$\tilde{\mathcal{M}} = (\boldsymbol{c}_1 \times \boldsymbol{c}_2, \boldsymbol{c}_1 \times \boldsymbol{c}_3, \boldsymbol{c}_1 \times \boldsymbol{c}_4, \boldsymbol{c}_2 \times \boldsymbol{c}_3, \boldsymbol{c}_2 \times \boldsymbol{c}_4, \boldsymbol{c}_3 \times \boldsymbol{c}_4),$$

where the vectors $\boldsymbol{c}_i$ $(i = 1, \ldots, 4)$ are the columns of $\mathcal{M}$. We can thus estimate the vectors $\boldsymbol{c}_i$ $(i = 1, 2, 3, 4)$ using constraints such as

$$\begin{pmatrix} \tilde{\boldsymbol{c}}_{12}^T \\ \tilde{\boldsymbol{c}}_{13}^T \\ \tilde{\boldsymbol{c}}_{14}^T \end{pmatrix} \boldsymbol{c}_1 = 0, \quad \tilde{\boldsymbol{c}}_{12} \cdot \boldsymbol{c}_3 = \tilde{\boldsymbol{c}}_{23} \cdot \boldsymbol{c}_1,$$

where $\tilde{\boldsymbol{c}}_{ij}$ denotes the values of $\boldsymbol{c}_i \times \boldsymbol{c}_j$ $(1 \leq i < j \leq 4)$ stored in the columns of $\tilde{\mathcal{M}}$. Collecting the 20 different equations of this type obtained by permuting the appropriate subscripts yields a systems of linear equations in the coordinates of the vectors $\boldsymbol{c}_i$ that can be solved once again using linear least squares (at most 11 of these 20 equations are of course linearly independent in the noise-free case).

We leave it to the reader to characterize the degenerate line configurations for which this method fails.

5.3.4 Analytical Photogrammetry

We present in this section a non-linear approach to camera calibration that takes into account all the constraints associated with a camera. This approach is borrowed from *photogrammetry*, an engineering field whose aim is to recover quantitative geometric information from one or several pictures, with applications in cartography, military intelligence, city planning, etc. [**?**; **?**]. For many years, photogrammetry relied on a combination of geometric, optical, and mechanical methods to recover three-dimensional information from pictures, but the advent of computers has made a purely computational approach to this problem feasible. This is the domain of *analytical photogrammetry*, where the intrinsic parameters of a camera define its *interior orientation*, while the extrinsinc parameters define its *exterior orientation.*

Let us write again the perspective projection equation as

$$\begin{cases} u = \dfrac{\boldsymbol{m}_1 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}}, \\ v = \dfrac{\boldsymbol{m}_2 \cdot \boldsymbol{P}}{\boldsymbol{m}_3 \cdot \boldsymbol{P}}. \end{cases}$$

Let $\boldsymbol{\xi}$ denote the vector formed by all intrinsic and extrinsic parameters of a camera. We can explicitly parameterize the projection matrix $\mathcal{M}$ and its columns $\boldsymbol{m}_i^T$ $(i = 1, 2, 3)$ by the vector $\boldsymbol{\xi}$ as in (5.2.7). In this setting, the problem of calibrating the camera reduces to minimizing the least-squares error

$$E(\boldsymbol{\xi}) = \sum_{i=1}^{n} [(u_i - \frac{\boldsymbol{m}_1(\boldsymbol{\xi}) \cdot \boldsymbol{P}_i}{\boldsymbol{m}_3(\boldsymbol{\xi}) \cdot \boldsymbol{P}_i})^2 + (v_i - \frac{\boldsymbol{m}_2(\boldsymbol{\xi}) \cdot \boldsymbol{P}_i}{\boldsymbol{m}_3(\boldsymbol{\xi}) \cdot \boldsymbol{P}_i})^2] \tag{5.3.14}$$

with respect to the coefficients $\boldsymbol{\xi}$.

Contrary to the cases studied so far, the dependency of each error term on the unknown parameters $\boldsymbol{\xi}$ is not linear. Instead, this dependency involves a combination of polynomial and trigonometric functions, and minimizing the overall error measure involves the use of *non-linear least squares* algorithms, as briefly discussed in the insert next page. These methods rely on the derivatives of the error function with respect to the unknown parameters to linearize the steps of the iterative minimization process. Non-linear least-squares techniques provide the means of computing a local minimum of E, and, when started with an appropriate initial guess, they will yield the global minimum as well.

Technique: Non-Linear Least Squares Methods

Let us consider a system of n non-linear equations in p unknowns:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_p) = b_1, \\ f_2(x_1, x_2, \ldots, x_p) = b_2, \\ \ldots \\ f_n(x_1, x_2, \ldots, x_p) = b_n, \end{cases} \tag{5.3.15}$$

where f_i denotes, for $i = 1, \ldots, n$, a differentiable function from $\mathbb{R}^p$ to $\mathbb{R}$. When

$$f_i(x_1, x_2, \ldots, x_p) = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ip}x_p,$$

we have of course exactly the same situation as in (5.3.2). In the general setting of *non-linear least squares*, the functions f_i can be arbitrarily non-linear. This time, we have (in general):

1. when $n < p$, there exists an $(p-n)$-dimensional *subspace of $\mathbb{R}^p$* formed by the vectors $\boldsymbol{x}$ that are solutions of (5.3.15);
2. when $n = p$, there exists a *finite set* of solutions;
3. when $n > p$, there is no solution.

We have emphasized in this statement the main differences with the linear case: the dimension of the solution set will still be $p - n$ (in general) in the underconstrained case, but this set will not form a vector space anymore. Its structure will depend on the nature of the functions f_i. Likewise, there will be (in general) a finite number of solutions instead of a unique one in the case $n = p$. This time we will not go into the details of what it means for a family of functions f_i $(i = 1, \ldots, n)$ to satisfy the "general" conditions under which the above statement is true. Linear and polynomial functions, for example, do satisfy these conditions.

From now on we will consider the overconstrained case $n > p$ and minimize the error

$$E(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{i=1}^{n} (f_i(\boldsymbol{x}) - b_i)^2.$$

The error function $E : \mathbb{R}^p \to \mathbb{R}^+$ may have many local minima, but it has (in general) a single global minimum. Except for the case of polynomial functions [**?**], there is unfortunately no numerical method guaranteed to find this global minimum. Effective iterative methods (e.g., gradient descent) do exist for finding a local minimum of a non-linear function, and it can be hoped that these methods will find the global minimum when they start from a reasonable initial guess. We will give in the remaining of this note such a method specialized for non-linear least squares.

The idea is to linearize the process: indeed, any smooth function looks like its tangent in a (small) neighborhood of any of its points. Formally, this can be rewritten using a first-order Taylor expansion as

$$f_i(\boldsymbol{x} + \delta\boldsymbol{x}) = f_i(\boldsymbol{x}) + \delta x_1 \frac{\partial f_i}{\partial x_1}(\boldsymbol{x}) + \ldots + +\delta x_p \frac{\partial f_i}{\partial x_p}(\boldsymbol{x}) + O(|\delta\boldsymbol{x}|^2) \approx f_i(\boldsymbol{x}) + \nabla f_i(\boldsymbol{x}) \cdot \delta\boldsymbol{x}, \tag{5.3.16}$$

where $\nabla f_i(\boldsymbol{x}) \stackrel{\text{def}}{=} (\frac{\partial f_i}{\partial x_1}, \dots, \frac{\partial f_i}{\partial x_p})^T$ is called the gradient of f_i at the point $\boldsymbol{x}$, and we have neglected the second-order term $O(|\delta \boldsymbol{x}|^2)$.

In the context of an iterative process, consider the problem of minimizing $E(\boldsymbol{x} + \delta \boldsymbol{x})$ with respect to $\delta \boldsymbol{x}$ for a given value of $\boldsymbol{x}$. Substituting (5.3.16) into (5.3.15) yields

$$E(\boldsymbol{x} + \delta\boldsymbol{x}) = \sum_{i=1}^{n} (f_i(\boldsymbol{x}) + \nabla f_i(\boldsymbol{x}) \cdot \delta\boldsymbol{x} - b_i)^2 = |\mathcal{J} \delta\boldsymbol{x} - \boldsymbol{c}|^2,$$

where

$$\mathcal{J} \stackrel{\text{def}}{=} \begin{pmatrix} \nabla f_1(\boldsymbol{x})^t \\ \dots \\ \nabla f_n(\boldsymbol{x})^t \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{x_1}(\boldsymbol{x}) & \dots & \frac{\partial f_1}{x_p}(\boldsymbol{x}) \\ \dots & \dots & \dots \\ \frac{\partial f_n}{x_1}(\boldsymbol{x}) & \dots & \frac{\partial f_n}{x_p}(\boldsymbol{x}) \end{pmatrix} \text{ and } \boldsymbol{c} \stackrel{\text{def}}{=} \begin{pmatrix} b_1 \\ \dots \\ b_n \end{pmatrix} - \begin{pmatrix} f_1(\boldsymbol{x}) \\ \dots \\ f_n(\boldsymbol{x}) \end{pmatrix}.$$

At this point we are back in the linear least-squares setting, and the adjustment $\delta \boldsymbol{x}$ can be computed as $\delta \boldsymbol{x} = \mathcal{J}^\dagger \boldsymbol{c}$. In practice the process is started with some initial guess $\boldsymbol{x}_0$, and a few iterations of the form $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \delta \boldsymbol{x}$ are sufficient to find a local (and hopefully global) minimum of E. Variants of this method include the Levenberg-Marquardt algorithm, popular in computer vision circles.

In our setting, we must compute the derivatives of the image coordinates with respect to the camera parameters. If $\boldsymbol{p} = \mathcal{M}\boldsymbol{P}$ and $\boldsymbol{p} = (p, q, r)^T$, we have

$$\begin{cases} \dfrac{\partial u}{\partial \xi} = \dfrac{\partial}{\partial \xi}(\dfrac{p}{r}) = \dfrac{1}{r}\dfrac{\partial p}{\partial \xi} - \dfrac{p}{r^2}\dfrac{\partial r}{\partial \xi} = \dfrac{1}{r}(\dfrac{\partial}{\partial \xi}(\boldsymbol{m}_1 \cdot \boldsymbol{P}) - u\dfrac{\partial}{\partial \xi}(\boldsymbol{m}_3 \cdot \boldsymbol{P})), \\ \dfrac{\partial v}{\partial \xi} = \dfrac{\partial}{\partial \xi}(\dfrac{q}{r}) = \dfrac{1}{r}\dfrac{\partial q}{\partial \xi} - \dfrac{q}{r^2}\dfrac{\partial r}{\partial \xi} = \dfrac{1}{r}(\dfrac{\partial}{\partial \xi}(\boldsymbol{m}_2 \cdot \boldsymbol{P}) - v\dfrac{\partial}{\partial \xi}(\boldsymbol{m}_3 \cdot \boldsymbol{P})), \end{cases}$$

which is easily rewritten as

$$\begin{pmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial v}{\partial \xi} \end{pmatrix} = \frac{1}{r}\begin{pmatrix} 1 & 0 & -u \\ 0 & 1 & -v \end{pmatrix} \frac{\partial \mathcal{M}}{\partial \xi}\boldsymbol{P}.$$

Note that u, v, r and $\boldsymbol{P}$ depend on the image considered, but that $\partial\mathcal{M}/\partial\xi$ only depends on the intrinsic and extrinsic parameters of the camera. Note also that this method requires an explicit parameterization of the matrix $\mathcal{R}$. Such a parameterization in terms of three elementary rotations about coordinate axes was mentioned earlier. Many other parameterizations are possible as well (Euler angles, matrix exponentials, quaternions, etc.), see [**?**; **?**; **?**] for discussions.

5.4 Notes

The book by Craig [**?**] offers a very good introduction to coordinate system representations and kinematics. A thorough presentation of camera models and the associated calibration methods can be found in the excellent book by Faugeras [**?**]. The calibration technique for taking radial distortion into account is adapted from Tsai [**?**]. The line-based calibration technique is inspired by the characterization of line projections in terms of the exterior product introduced by Faugeras and Papadopoulo [**?**]. The book of Haralick and Shapiro [**?**] presents an excellent concise introduction to analytical photogrammetry. The *Manual of Photogrammetry* is of course the gold standard, and newcomers to this field (like the authors of this book) will probably find the ingenious mechanisms and rigorous methods described in the various editions of this book fascinating [**?**; **?**]. We will come back to photogrammetry in the context of multiple images in Chapter 11.

5.5 Assignments

Exercises

1. Write formulas for the matrices ${}^A_B\mathcal{R}$ when (B) is deduced from (A) via a rotation of angle θ about the axes $\boldsymbol{i_A}$, $\boldsymbol{j}_A$ and $\boldsymbol{k}_A$ respectively.

2. Show that rotation matrices are characterized by the following properties: (1) the inverse of a rotation matrix is equal to its transpose, and (2) its determinant is equal to 1.

3. Show that the set of matrices associated with rigid transformations and equipped with the matrix product forms a group.

4. Let ${}^A\mathcal{T}$ denote the matrix associated with a rigid transformation $\mathcal{T}$ in the coordinate system (A), with

$$ {}^A\mathcal{T} = \begin{pmatrix} {}^A\mathcal{R} & {}^A\boldsymbol{T} \\ \boldsymbol{0} & 1 \end{pmatrix}. $$

   Construct the matrix ${}^B\mathcal{T}$ associated with $\mathcal{T}$ in the coordinate system (B) as a function of ${}^A\mathcal{T}$ and the rigid transformation separating (A) and (B).

5. Show that if the coordinate system (B) is obtained by applying to the coordinate system (A) the transformation associated with the 4×4 matrix $\mathcal{T}$, then ${}^BP = \mathcal{T}^{-1}\,{}^AP$.

6. Show that the rotation of angle θ about the $\boldsymbol{k}$ axis of the frame (F) can be represented by

$$ {}^FP' = \mathcal{R} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} {}^FP. $$

7. Show that the change of coordinates associated with a rigid transformation preserves distances and angles.

8. Show that when the camera coordinate system is skewed and the angle θ between the two image axes is not equal to 90 degrees, then (5.2.3) transforms into (5.2.4).

9. Let $\boldsymbol{C}$ denote the coordinate vector of the optical center of a camera in some reference frame, and let $\mathcal{M}$ denote the corresponding perspective projection matrix. Show that

$$ \mathcal{M} \begin{pmatrix} \boldsymbol{C} \\ 1 \end{pmatrix} = 0. $$

10. Show that the conditions of Theorem 2 are necessary.

11. Show that the conditions of Theorem 2 are sufficient. Note that the statement of this theorem is a bit different from the corresponding theorems in [**?**; **?**], where the condition $\mathrm{Det}(\mathcal{A}) \neq 0$ is replaced by $\boldsymbol{a}_3 \neq 0$. But of course $\mathrm{Det}(\mathcal{A}) \neq 0$ implies $\boldsymbol{a}_3 \neq 0$.

12. Consider some coordinate system and the Plücker coordinate vector $\boldsymbol{\Delta}$ of the line Δ passing through the points A and B. Show that if O denotes the origin of the coordinate system and H denotes its projection onto Δ, then $\overrightarrow{AB} = -(\Delta_3, \Delta_5, \Delta_6)$ and $\overrightarrow{OA} \times \overrightarrow{OB} = \overrightarrow{OH} \times \overrightarrow{AB} = (\Delta_4, -\Delta_2, \Delta_1)^T$.

13. Show analytically that the following identity holds for any 4-vectors $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$:
$$(\boldsymbol{a} \wedge \boldsymbol{b}) \cdot (\boldsymbol{c} \wedge \boldsymbol{d}) = (\boldsymbol{a} \cdot \boldsymbol{c})(\boldsymbol{b} \cdot \boldsymbol{d}) - (\boldsymbol{a} \cdot \boldsymbol{d})(\boldsymbol{b} \cdot \boldsymbol{c}).$$

14. Show that
$$\tilde{\mathcal{M}} = (\boldsymbol{c}_1 \times \boldsymbol{c}_2, \boldsymbol{c}_1 \times \boldsymbol{c}_3, \boldsymbol{c}_1 \times \boldsymbol{c}_4, \boldsymbol{c}_2 \times \boldsymbol{c}_3, \boldsymbol{c}_2 \times \boldsymbol{c}_4, \boldsymbol{c}_3 \times \boldsymbol{c}_4) \quad \text{where} \quad \mathcal{M} = (\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3, \boldsymbol{c}_4).$$

Programming Assignments

Note: the assignments below require routines for solving square and overdetermined linear systems. An extensive set of such routines is available in MATLAB as well as in public-domain libraries such as LINPACK and LAPACK that can be downloaded from the Netlib repository (`http://www.netlib.org/`). Data for these assignments will be available in the CD companion to this book.

1. Use linear least-squares to fit a plane to n data points $(x_i, y_i, z_i)^T$ $(i = 1, \ldots, n)$ in $\mathbb{R}^3$.

2. Use linear least-squares to fit a conic section defined by
$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$
to n data points $(x_i, y_i)^T$ $(i = 1, \ldots, n)$ in $\mathbb{R}^2$.

Chapter 6

AN INTRODUCTION TO PROBABILITY

As the previous chapters have illustrated, it is often quite easy to come up with physical models that determine the effects that result from various causes — we know how image intensity is determined, for example. The difficulty is that effects could have come from various causes and we would like to know which — for example, is the image dark because the light level is low, or because the surface has low albedo? Ideally, we should like to take our measurements and determine a reasonable description of the world that generated them. Accounting for uncertainty is a crucial component of this process, because of the ambiguity of our measurements. This process of accountancy needs to take into account reasonable preferences about the state of the world — for example, it is less common to see very dark surfaces under very bright lights than it is to see a range of albedoes under a reasonably bright light.

Probability is the proper mechanism for accounting for uncertainty. Axiomatic probability theory is gloriously complicated, and we don't attempt to derive the ideas in detail. Instead, this chapter will first review the basic ideas of probability. We then describe techniques for building probabilistic models and for extracting information from a probabilistic model, all in the context of quite simple examples. In chapter **??**, we show some substantial examples of probabilistic methods; there are other examples scattered about the text by topic.

Discussions of probability are often bogged down with waffle about what probability *means*, a topic that has attracted a spectacular quantity of text. Instead, we will discuss probability as a modelling technique with certain formal, abstract properties — this means we can dodge the question of what the ideas mean and concentrate on the far more interesting question of what they can do for us.

We will develop probability theory in discrete spaces first, because it is possible to demonstrate the underpinning notions without much notation. We then pass to continuous spaces.

6.1 Probability in Discrete Spaces

Generally, a probability model is used to compare various kinds of experimental outcome that can be distinguished. These outcomes are usually called **events**. Now if it is possible to tell whether an event has occurred, it is possible to tell if it has not occurred, too. Furthermore, if it is possible to tell that two events have occurred independently, then it is possible to tell if they have occurred simultaneously.

This motivates a formal structure. We take a discrete space, D, which could be infinite and which represents the world in which experiments occur. Now construct a collection of subsets of D, which we shall call $\mathcal{F}$, with the following properties:

- The empty set is in $\mathcal{F}$ and so is D.
- *Closure under intersection:* if $S_1 \in \mathcal{F}$ and $S_2 \in \mathcal{F}$, then $S_1 \cap S_2 \in \mathcal{F}$.
- *Closure under complements:* if $S_1 \in \mathcal{F}$ then $\overline{S_1} = D - S_1 \in \mathcal{F}$.

The elements of $\mathcal{F}$ correspond to the events. Note that we can we can tell whether any logical combinations of events has occurred, too, because a logical combination of events corresponds to set unions, negations or intersections.

Given a coin that is flipped once,

$$D = \{\texttt{heads}, \texttt{tails}\}$$

There are only two possible sets of events in this case:

$$\{\emptyset, D\}$$

(which implies we flipped the coin, but can't tell what happened!) and

$$\{\emptyset, D, \{\texttt{heads}\}, \{\texttt{tails}\}\}$$

Example 6.1: *The space of events for a single toss of a coin.*

6.1.1 Probability: the P-function

Now we construct a function P, which takes elements of $\mathcal{F}$ to the unit interval. We require that P has some important properties:

- P is defined for every element of $\mathcal{F}$
- $P(\emptyset) = 0$
- $P(D) = 1$
- for $A \in \mathcal{F}$ and $B \in \mathcal{F}$, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Given two coins that are flipped,

$$D = \{\texttt{hh}, \texttt{ht}, \texttt{tt}, \texttt{th}\}$$

There are rather more possible sets of events in this case. One useful one would be

$$\mathcal{F} = \left\{ \begin{array}{cccc} \emptyset, & D, & & \\ \{\texttt{hh}\}, & \{\texttt{ht}\}, & \{\texttt{tt}\}, & \{\texttt{th}\}, \\ \{\texttt{hh}, \texttt{ht}\}, & \{\texttt{hh}, \texttt{th}\}, & \{\texttt{hh}, \texttt{tt}\}, & \{\texttt{ht}, \texttt{th}\}, \\ \{\texttt{ht}, \texttt{tt}\}, & \{\texttt{th}, \texttt{tt}\}, & \{\texttt{hh}, \texttt{ht}, \texttt{th}\}, & \{\texttt{hh}, \texttt{ht}, \texttt{tt}\}, \\ \{\texttt{hh}, \texttt{th}, \texttt{tt}\}, & \{\texttt{ht}, \texttt{th}, \texttt{tt}\}\} & & \end{array} \right\}$$

which would correspond to all possible cases. Another (perhaps less useful) structure would be:

$$\mathcal{F} = \{\emptyset, D, \{\texttt{hh}, \texttt{ht}\}, \{\texttt{th}, \texttt{tt}\}\}\}$$

which implies that we cannot measure the state of the second coin

Example 6.2: *Two possible spaces of events for a single flip each of two coins.*

which we call *the axiomatic properties of probability.* Note that $0 \leq P(A) \leq 1$ for all $A \in \mathcal{F}$, because the function takes elements of $\mathcal{F}$ to the unit interval. We call the collection of D, P and $\mathcal{F}$ a **probability model**. We call $P(A)$ the **probability of the event A** — because we are still talking about formal structures, there is absolutely no reason to discuss what this means; it's just a name. Rigorously justifying the properties of P is somewhat tricky — Jaynes' book ([]) is one place to start, as is []. It can be helpful to think of P as a function that measures the size of a subset of D — the whole of D has size one, and the size of the union of two disjoint sets is the sum of their sizes.

In example 1, for the first structure on D, there is only one possible choice of P; for the second, there is a one parameter family of choices, we could choose $P(\texttt{heads})$ to be an arbitrary number in the unit interval, and the choice of $P(\texttt{tails})$ follows.

Example 6.3: *The possible P functions for the flip of a single coin.*

In example 2, there is a three-parameter family of choices for P in the case of the first event structure shown in that example — we can choose $P(\texttt{hh})$, $P(\texttt{ht})$ and $P(\texttt{th})$, and all other values will be given by the axioms. For the second event structure in that example, P is the same as that for a single coin (because we can't tell the state of one coin).

Example 6.4: *The P functions for two coins, each flipped once.*

6.1.2 Conditional Probability

If we have some element A of $\mathcal{F}$ where $P(A) \neq 0$ — and this constraint is important — then the collection of sets

$$\mathcal{F}_{\mathcal{A}} = \{u \cap A | u \in \mathcal{F}\}$$

has the same properties as $\mathcal{F}$. Furthermore, the function with domain $\mathcal{F}_A$

$$P_A(C) = \frac{P(C)}{P(A)}$$

(where $C \in \mathcal{F}_A$) also satisfies the axiomatic properties of probability on its domain. We call this function **the conditional probability of C, given A**; it is usually written as $P(C|A)$. If we adopt the metaphor that P measures the size of a set, then the conditional probability measures the size of the set $C \cap A$ relative to A. Notice that

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

an important fact that you should memorize.

Assume that we have a collection of n sets A_i, such that $A_j \cap A_k = \emptyset$ for every $j \neq k$ and $A = \bigcup_i A_i$. The analogy between probability and size motivates the result that

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

a fact well worth remembering.

6.1.3 Choosing P

We have a formal structure — to use it, we need to choose values of P that have useful semantics. There are a variety of ways of doing this, and it is essential to understand that there is no canonical choice. The choice of P is an essential part of the modelling process. A bad choice will lead to an unhelpful or misleading model, and a good choice may lead to a very enlightening model. There are some strategies that help in choosing P.

Symmetry

Many problems have a form of symmetry that means we have no reason to distinguish between certain sets of events. In this case, it is natural to choose P to reflect this fact.

Assume we have a single coin which we will flip, and we can tell the difference between heads and tails. Then

$$\mathcal{F} = \{\emptyset, D, \{\texttt{heads}\}, \{\texttt{tails}\}\}$$

is a reasonable model to adopt. Now this coin is symmetric — there is no reason to distinguish between the heads side and the tails side from a mechanical perspective. Furthermore, the operation of flipping it subjects it to mechanical forces that do not favour one side over the other. In this case, we have no reason to believe that there is any difference between the outcomes, so it is natural to choose

$$P(\texttt{heads}) = P(\texttt{tails})$$

Example 6.5: *Choosing the P function for a single coin flip using symmetry.*

Assume we have a die that we believe to be fair, in the sense that it has been manufactured to have the symmetries of a cube. A symmetry argument allows us to assume that the probability that each face comes up is equal because we have no reason to prefer faces.

Example 6.6: *Choosing the P function for a roll of a die using symmetry.*

Independence

In many probability models, events do not depend on one another. This is reflected in the conditional probability. If there is no interaction between events A and B, then $P(A|B)$ cannot depend on B. This means that $P(A|B) = P(A)$, a property known as **independence**. In turn, if A and B are independent, we have $P(A \cap B) = P(A|B)P(B) = P(A)P(B)$. This property is important, because it reduces the number of parameters that must be chosen in building a probability model.

A more subtle version of this property is **conditional independence**. Formally, A and B are conditionally independent given C if

$$P(A, B, C) = P(A, B|C)P(C) = P(A|C)P(B|C)P(C)$$

Like independence, conditional independence simplifies modelling by (sometimes substantially) reducing the number of parameters that must be chosen in constructing a model.

We adopt the first of the two event structures given for the two coins in example 2 (this is where we can tell the state of both coins). Now we assume that neither coin knows the other's intentions or outcome.
This assumption restricts our choice of probability model quite considerably because it enforces a symmetry. Let us choose

$$P(\{\texttt{hh}, \texttt{ht}\}) = p_{1h}$$

and

$$P(\{\texttt{hh}, \texttt{th}\}) = p_{2h}$$

Now let us consider conditional probabilities, in particular

$$P(\{\texttt{hh}, \texttt{ht}\}|\{\texttt{hh}, \texttt{th}\})$$

(which we could interpret as the probability that the first coin comes up heads given the second coin came up heads). If the coins cannot communicate, then this conditional probability should not depend on the conditioning set, which means that

$$P(\{\texttt{hh}, \texttt{ht}\}|\{\texttt{hh}, \texttt{th}\}) = P(\{\texttt{hh}, \texttt{ht}\})$$

In this case, we know that

$$P(\texttt{hh}) = P(\{\texttt{hh}, \texttt{ht}\}|\{\texttt{hh}, \texttt{th}\})P(\{\texttt{hh}, \texttt{th}\}) = P(\{\texttt{hh}, \texttt{ht}\})P(\{\texttt{hh}, \texttt{th}\}) = p_{1h}p_{2h}$$

Similar reasoning yields $P(A)$ for all $A \in \mathcal{F}$, so that our assumption that the two coins are independent means that there is now only a two parameter family of probability models to choose from — one parameter describes the first coin, the other describes the second.

Example 6.7: *Choosing the P function for a single flip each of two coins using the idea of independence.*

Frequency:

Data reflecting the relative frequency of events can be easily converted into a form that satisfies the axioms for P, as example 9 indicates.

An interpretation of probability as frequency is consistent, in the sense that if we make repeated, independent trials of a probability model where P has been allocated using frequency data, then the events with the highest probability — which will be long sequences of outcomes — will be those that show the outcomes with about the right frequency. Example 10 illustrates this effect for repeated flips of a single coin.

Saying that the relative frequency of an event is f means that, in a very large number of trials (say, N), we expect that the event occurs in about fN of those

Both I and my neighbour have a lawn; each lawn has its own sprinkler system. There are two reasons that my lawn could be wet in the morning — either it rained in the night, or my sprinkler system came on. There is no reason to believe that the neighbour's sprinkler system comes on at the same times or on the same days as mine does. Neither sprinkler system is smart enough to know whether it has rained. Finally, if it rains, both lawns are guaranteed to get wet; however, if the sprinkler system comes on, there is some probability that the lawn will not get wet (perhaps a jammed nozzle).

A reasonable model has five binary variables (my lawn is wet or not; the neighbour's lawn is wet or not; my sprinkler came on or not; the neighbour's sprinkler came on or not; and it rained or not). D has 32 elements, and the event space is too large to write out conveniently. If there was no independence in the model, specifying P could require 31 parameters.

However, if I know whether it rained in the night, then the state of my lawn is independent of the state of the neighbour's lawn. Our joint probability function is

$$P(\mathtt{W}, \mathtt{W}_n, \mathtt{S}, \mathtt{S}_n, \mathtt{R}) \;=\; P(\mathtt{W}, \mathtt{S}|\mathtt{R})P(\mathtt{W}_n, \mathtt{S}_n|\mathtt{R})P(\mathtt{R})$$

We know that $P(\mathtt{W} = \text{true}, \mathtt{S}|\mathtt{R} = \textit{true}) = P(\mathtt{S})$ (this just says that if it rains, the lawn is going to be wet); a similar observation applies to the neighbour's lawn. The rain and the sprinklers are independent *and* there is a symmetry — both my neighbour's lawn and mine behave in the same. This means that, in total, we need only 5 parameters to specify this model.

Example 6.8: *Simplifying a model using conditional independence: the case of rain, sprinklers and lawns.*

Assume that, in the past, we have flipped the single coin described above many times, and observed that for 51% of these flips it comes up heads, and for 49% it comes up tails. We could choose

$$P(heads) = 0.51 \text{ and } P(tails) = 0.49$$

This choice is a sensible choice, as example 10 indicates.

Example 6.9: *Choosing a P function for a single coin flip using frequency information.*

trials. Now for large n, the expression

$$\left(\begin{array}{c} k \\ n \end{array} \right) p^k (1-p)^{n-k}$$

Now consider a single coin that we flip many times, and where each flip is independent of the other. We set up an event structure that does not reflect the order in which the flips occur. For example, for two flips, we would have:

$$\{\emptyset, D, \texttt{hh}, \texttt{tt}, \{\texttt{ht}, \texttt{th}\}, \{\texttt{hh}, \texttt{tt}\}, \{\texttt{hh}, \texttt{ht}, \texttt{th}\}, \{\texttt{tt}, \texttt{ht}, \texttt{tt}\}\}$$

We assume that $P(\texttt{hh}) = p^2$; a simple computation using the idea of independence yields that $P(\{\texttt{ht}, \texttt{th}\}) = 2p(1-p)$ and $P(\texttt{tt}) = (1-p)^2$. We can generalise this result, to obtain

$$P(k \text{ heads and } n-k \text{ tails in } n \text{ flips}) = \left(\begin{array}{c} k \\ n \end{array} \right) p^k (1-p)^{n-k}$$

Example 6.10: *The probability of various frequencies in repeated coin flips*

(which is what we obtained for the probability of a sequence of trials showing k heads and $n-k$ tails in example 10) has a substantial peak at $p = \frac{k}{n}$. This peak gets very narrow and extremely pronounced as $n \to \infty$. This effect is extremely important, and is consistent with an interpretation of probability as relative frequency:

- firstly, because it means that we assign a high probability to long sequences of coin flips where the event occurs with the “right” frequency
- and secondly, because the probability assigned to these long sequences can also be interpreted as a frequency — essentially, this interpretation means that long sequences where the events occur with the “right” frequency occur far more often than other such sequences (see figure 6.1).

All this means that, if we choose a P function for a coin flip — or some other experiment — on the basis of *sufficiently good* frequency data, then we are very unlikely to see long sequences of coin flips — or repetitions of the experiment — that do not show this frequency.

This interpretation of probability as frequency is widespread, and common. One valuable advantage of the interpretation is that it simplifies estimating probabilities for some sorts of models. For example, given a coin, one could obtain $P(\texttt{heads})$ by flipping the coin many times and measuring the relative frequency with which heads appear.

Subjective probability

It is not always possible to use frequencies to obtain probabilities. There are circumstances in which we would like to account for uncertainty but cannot meaningfully speak about frequencies. For example, it is easy to talk about the probability it will rain tomorrow, but hard to interpret this use of the term as a statement about

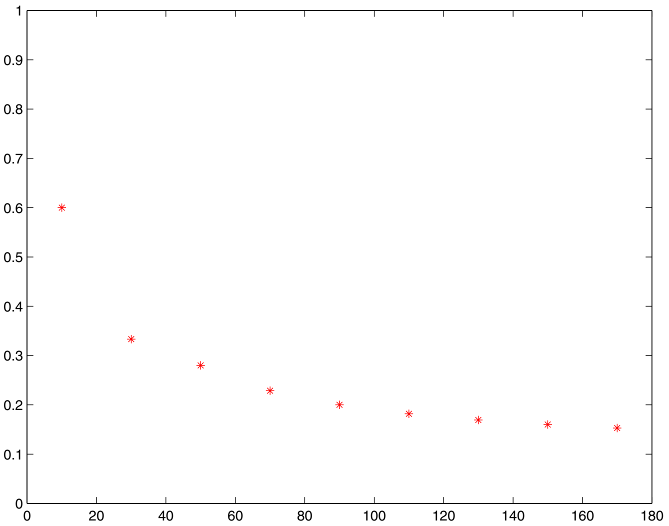


Figure 6.1. We assume that a single flip of a coin has a probability 0.5 of coming up heads. If we interpret probability as frequency, then long sequences of coin flips should almost always have heads appearing about half the time. This plot shows the width of the interval about 0.5 that contains 95% of the probability for various numbers of repeated coin flips. Notice that as the sequence gets longer, the interval gets narrower — one is very likely to observe a frequency of heads in the range $[0.43, 0.57]$ for 170 flips of a coin of this kind.

frequency[1]. An alternative source of P is to regard probability as encoding *degree of belief*. In this approach, which is usually known as **subjective probability**, one chooses P to reflect reasonable beliefs about the situation that applies.

Subjective probability must still satisfy the axioms of probability. It is simply a way of choosing free parameters in a probability model without reference to frequency. The attractive feature of subjective probability is that it emphasizes that a choice of probability model is a *modelling* exercise — there are few circumstances where the choice is canonical. One natural technique to adopt is to choose a function P that yields good behaviour in practice, an approach known as **learning** and discussed in chapter **??**.

---

[1] One dodge is to assume that there are a very large set of equivalent universes which are the same today. In some of these worlds, it rains tomorrow and in others it doesn't; the frequency with which it rains tomorrow is the probability. This philosophical fiddle isn't very helpful in practice, because we can't actually measure that frequency by looking at these alternative worlds.

A friend with a good reputation for probity and no obvious need for money draws a coin from a pocket, and offers to bet with you on whether it comes up heads or tails — your choice of face. What probability do you ascribe to the event that it comes up heads?
Now an acquaintance draws a coin from a pocket and offers a bet: he'll pay you 15 dollars for your stake of one dollar if the coin comes up heads. What probability do you ascribe to the event that it comes up heads?
Finally you encounter someone in a bar who (it emerges) has a long history of disreputable behaviour and an impressive conviction record. This person produces a coin from a pocket and offers a bet: you pay him 1000 dollars for his stake of one dollar if it lands on its edge and stands there. What probability do you ascribe to the event that it lands on its edge and stands there?
You have to choose your answer for these cases — that's why it's subjective — but you could lose a lot of money learning that the answer in the second case is going to be pretty close to zero and in the third case is pretty close to one.

Example 6.11: *Assigning P functions to two coins from two different sources, using subjective probability.*

6.2 Probability in Continuous Spaces

Much of the discussion above transfers quite easily to a continuous space, as long as we are careful about events. The difficulty is caused by the "size" of continuous spaces — there are an awful lot of numbers between 1.0 and 1.00000001, one for each number between 1.0 and 2.0. For example, if we are observing noise — perhaps by measuring the voltage across the terminals of a warm resistor — the noise will very seldom take the value 1 exactly. It is much more helpful to consider the probability that the value is in the range 1 to $1 + \delta$, for δ a small step.

6.2.1 Event Structures for Continuous Spaces

This observation justifies using events that look like intervals or boxes for continuous spaces. Given a space D, our space of events will be a set $\mathcal{F}$ with the following properties:

- The empty set is in $\mathcal{F}$ and so is D.
- *Closure under finite intersections:* if S_i is a *finite collection* of subsets, and each $S_i \in \mathcal{F}$ then $\cap_i S_i \in \mathcal{F}$.
- *Closure under finite unions:* if S_i is an *finite collection* of subsets, and each $S_i \in \mathcal{F}$ then $\cup_i S_i \in \mathcal{F}$.
- *Closure under complements:* if $S_1 \in \mathcal{F}$ then $\overline{S_1} = D - S_1 \in \mathcal{F}$.

The basic axioms for P apply here too. For D the domain, and A and B events, we have:

- $P(D) = 1$
- $P(\emptyset) = 0$
- for any A, $0 \leq P(A) \leq 1$
- if $A \subset B$, then $P(A) \leq P(B)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

The concepts of conditional probability, independence and conditional independence apply in continuous spaces without modification. For example, the conditional probability of an event given another event can be defined by

$$P(A \cap B) = P(A|B)P(B)$$

and the conditional probability can be thought of as probability restricted to the set B. Events A and B are independent if and only if

$$P(A \cap B) = P(A)P(B)$$

and A and B are conditionally independent given C if and only if

$$P(A \cap B|C) = P(A|C)P(B|C)$$

The main difficulty is expressing the function P in a useful way — it is clearly no longer possible to write down the space of events and give a value of P for each event. We will deal only with R^n, with subsets of this space, or with multiple copies of this space.

6.2.2 Representing a P-function for the Real Line

The set of events for the real line is far too big to write down. All events look like unions of a basic collection of sets. This basic collection consists of:

- individual points (i.e a);
- open intervals (i.e. (a, b));
- half-open intervals (i.e. $(a, b]$ or $[a, b)$);
- and closed intervals (i.e. $[a, b]$).

All of these could extend to infinity. The function P can be represented by a function F with the following properties:

- $F(-\infty) = 0$

- $F(\infty) = 1$
- $F(x)$ is monotonically increasing.

and we interpret $F(x)$ as $P((-\infty, x])$. The function F is referred to as the **cumulative distribution function**.

The value of P for all the basic sets described can be extracted from F, with appropriate attention to limits; for example, $P((a, b]) = F(b) - F(a)$ and $P(a) = \lim_{\epsilon \leftarrow 0^+}(F(a+\epsilon) - F(a))$. Notice that if F is continuous, $P(a) = 0$.

6.2.3 Probability Densities

In R^n, events are unions of elements of a basic collection of sets, too. This basic collection consists of a product of n elements from the basic collection for the real line. A cumulative distribution function can be defined in this case, too. It is given by a function F with the property that $P(\{x_1 \leq u_1, x_2 \leq u_2, \ldots x_n \leq u_n\}) = F(\boldsymbol{u})$. This function is constrained by other properties, too. However, cumulative distribution functions are a somewhat unwieldy way to specify probability.

For the examples we will deal with in continuous spaces, the usual way to specify P is to provide a function p such that

$$P(\text{event}) = \int_{\text{event}} p(u) du$$

This function is referred to as a **probability density function**. Not every probability model admits a density function, but all our cases will. Note that a density function cannot have a negative value, but that its value could be larger than one. In all cases, probability density functions integrate to one, i.e.

$$P(D) = \int_D p(u) du = 1$$

and any non-negative function with this property is a probability density function. The value of the probability density function at a point represents the probability of the event that consists of an infinitesimal neighbourhood at that value, i.e.:

$$p(u_1)du = P(\{u \in [u_1, u_1 + du]\})$$

Conditional probability, independence and conditional independence are ideas that can be translated into properties of probability density functions. In their most useful form, they are properties of random variables.

6.3 Random Variables

Assume that we have a probability model on either a discrete or a continuous domain, $\{D, \mathcal{F}, P\}$. Now let us consider a function of the outcome of an experiment. The values that this function takes on the different elements of D form a new set, which we shall call D'. There is a structure, with the same formal properties as $\mathcal{F}$ on D' defined by the values that this function takes on different elements of $\mathcal{F}$ — call this structure $\mathcal{F}'$.

This function is known as a **random variable**. We can talk about the probability that a random variable takes a particular set of values, because the probability structure carries over. In particular, assume that we have a random variable ξ. If $A' \in \mathcal{F}'$, there is some $A \in \mathcal{F}$ such that $A' = \xi(A)$. This means that

$$P(\{\xi \in A'\}) = P(A)$$

The simplest random variable is given by the identity function — this means that D' is the same as D, and $\mathcal{F}'$ is the same as $\mathcal{F}$. For example, the outcome of a coin flip is a random variable.

Now gamble on the outcome of a coin flip: if it comes up heads, you get a dollar, and if it comes up tails, you pay a dollar. Your income from this gamble is a random variable. In particular, $D' = \{1, -1\}$ and $\mathcal{F}' = \{\emptyset, D', \{1\}, \{-1\}\}$.

Now gamble on the outcome of two coin flips: if both coins come up the same, you get a dollar, and if they come up different, you pay a dollar. Your income from this gamble is a random variable. Again, $D' = \{1, -1\}$ and $\mathcal{F}' = \{\emptyset, D', \{1\}, \{-1\}\}$. In this case, D' is not the same as D and $\mathcal{F}'$ is not the same as $\mathcal{F}$; however, we can still speak about the probability of getting a dollar — which is the same as $P(\{hh, tt\})$.

Example 6.12: *The payoff on a gamble is a random variable.*

Density functions are very useful for specifying the probability model for the value of a random variable. However, they do result in quite curious notations (probability is a topic that seems to encourage creative use of notation). It is common to write the density function for a random variable as p. Thus, the distribution for λ would be written as $p(\lambda)$ — in this case, the name of the *variable* tells you what function is being referred to, rather than the name of the function, which is always p. Some authors resist this convention, but its use is pretty much universal in the vision literature, which is why we adopt it. For similar reasons, we write the probability function for a set of events as P, so that the probability of an event $P(\text{event})$ (despite the fact that different sets of events may have very different probability functions).

6.3.1 Conditional Probability and Independence

Conditional probability is a very useful idea for random variables. Assume we have two random variables, m and n — (for example, the value I read from my rain gauge as m and the value I read on the neighbour's as n). Generally, the probability density function is a function of both variables, $p(m, n)$. Now

$$p(m_1, n_1) dm dn = P(\{m \in [m_1, m_1 + dm]\} \text{ and } \{n \in [n_1, n_1 + dm]\})$$

$$= P(\{m \in [m_1, m_1 + dm]\} \,|\, \{n \in [n_1, n_1 + dm]\}) P(\{n \in [n_1, n_1 + dm]\})$$

We can define a conditional probability density from this by

$$\begin{aligned} p(m_1, n_1) dm dn &= P(\{m \in [m_1, m_1 + dm]\} \,|\, \{n \in [n_1, n_1 + dm]\}) P(\{n \in [n_1, n_1 + dm]\}) \\ &= (p(m_1|n_1) dm)(p(n_1) dn) \end{aligned}$$

Note that this conditional probability density has the expected property, that

$$p(m|n) = \frac{p(m, n)}{p(n)}$$

Independence and conditional independence carry over to random variables and probability densities without fuss.

We now consider the probability that each of two coins comes up heads, yielding two random variables — the relevant probabilities — which we shall write as p_1 and p_2. Now the density function for these random variables is $p(p_1, p_2)$. There is very little reason to believe that there is any dependency between these coins, so we should be able to write $p(p_1, p_2) = p(p_1)p(p_2)$. Notice that the notation is particularly confusing here; the intended meaning is that $p(p_1, p_2)$ factors, but that the factors are not necessarily equal. *In this case*, a further reasonable modelling step is to assume that $p(p_1)$ is the same function as $p(p_2)$

Example 6.13: *Independence in random variables associated with two coins.*

6.3.2 Expectations

The **expected value** or **expectation** of a random variable (or of some function of the random variable) is obtained by multiplying each value by its probability and summing the results — or, in the case of a continuous random variable, by multiplying by the probability density function and integrating. The operation is known as **taking an expectation**. For a discrete random variable, x, taking the expectation of x yields:

$$\mathrm{E}[x] = \sum_{i \in \text{values}} x_i p(x_i)$$

For a continuous random variable, the process yields

$$\mathrm{E}[x] = \int_D x p(x) dx$$

often referred to as the average, or the mean in polite circles. One model for an expectation is to consider the random variable as a payoff, and regard the expectation as the average reward, per bet, for an infinite number of repeated bets. The expectation of a general function $g(x)$ of a random variable x is written as $\mathrm{E}[g(x)]$.

The **variance** of a random variable x is

$$\mathrm{var}(x) = \mathrm{E}[x^2 - (\mathrm{E}(x))^2]$$

This expectation measures the average deviance from the mean. The variance of a random variable gives quite a strong indication of how common it is to see a value that is significantly different from the mean value. In particular, we have the following useful result:

$$P(\{\mid x - \mathrm{E}[x] \mid \geq \epsilon\}) \leq \frac{\mathrm{var}(x)}{\epsilon^2} \tag{6.3.1}$$

You and an acquaintance decide to bet on the outcome of a coin flip. You will receive a dollar from your acquaintance if the coin comes up heads, and pay one if it comes up tails. The coin is symmetric.
This means the expected value of the payoff is

$$1P(\texttt{heads}) - 1P(\texttt{tails}) = 0$$

The variance of the payoff is one, as is the standard deviation.
Now consider the probability of obtaining 10 dollars in 10 coin flips, with a fair coin. Our random variable x is the income in 10 coin flips. Equation 6.3.1 yields $P(\{\mid x \mid \geq 10\}) \leq \frac{1}{100}$, which is a generous upper bound — the actual probability is of the order of one in a thousand.

Example 6.14: *The expected value of gambling on a coin flip.*

The **standard deviation** is obtained from the variance:

$$\mathrm{sd}(x) = \sqrt{\mathrm{var}(x)} = \sqrt{\mathrm{E}[x^2 - (\mathrm{E}[x])^2]}$$

For a vector of random variables, the **covariance** is

$$\mathrm{cov}(\boldsymbol{x}) = \mathrm{E}[\boldsymbol{x}\boldsymbol{x}^t - (\mathrm{E}[x]\mathrm{E}[x]^t)]$$

This matrix (look carefully at the transpose) is symmetric. Diagonal entries are the variance of components of $\boldsymbol{x}$, and must be non-negative. Off-diagonal elements measure the extent to which two variables co-vary. For independent variables, the covariance must be zero. For two random variables that generally have different signs, the covariance can be negative.

Expectations of functions of random variables are extremely useful. The notation for expectations can be a bit confusing, because it is common to omit the density with respect to which the expectation is being taken, which is usually obvious from the context. For example, $\mathrm{E}[x^2]$ is interpreted as

$$\int_D x^2 p(x) dx$$

6.3.3 Joint Distributions and Marginalization

Assume we have a model describing the behaviour of a collection of random variables. We will proceed on the assumption that they are discrete, but (as should be clear by now) the discussion will work for continuous variables if summing is replaced by integration. One way to specify this model is to give the probability distribution for all variables, known in jargon as the **joint probability distribution function** — for concreteness, write this as $P(x_1, x_2, \ldots x_n)$. If the probability distribution is represented by its density function, the density function is usually referred to as the **joint probability density function**. Both terms are often abbreviated as "joint."

As we have already seen, the value of P for some elements of the event space can be determined from the value of P for other elements. This means that if we know

$$P(\{x_1 = a, x_2 = b, \ldots x_n = n\})$$

for each possible value of $a, b, \ldots, n$, then we should know P for a variety of other events. For example, it might be useful to know $P(\{x_1 = a\})$.

It should be obvious that the event structure, while useful, is getting unwieldy as a notation. It is quite common to use a rather sketchy notation to indicate the appropriate event. For example 15, we would write

$$P(\{(\texttt{heads}, I), (\texttt{heads}, II)\}) = P(\texttt{heads})$$

for example. In this notation, the argument of example 15 leads to:

$$P(x_2, \ldots x_n) = \sum_{\text{values of } x_1} P(x_1, x_2, \ldots x_n)$$

This operation is referred to as **marginalisation**.

A similar argument applies to probability density functions, but the operation is now integration. Given a probability density function $p(x_1, x_2, \ldots, x_n)$, we obtain

$$p(x_2, \ldots x_n) = \int_D p(x_1, x_2, \ldots x_n) dx_1$$

6.4 Standard Distributions and Densities

There are a variety of standard distributions that arise regularly in practice. References such as [] give large numbers; we will discuss only the most important cases.

The **uniform distribution** has the same value at each point on the domain. This distribution is often used to express an unwillingness to make a choice or a lack of information. On a continuous space, the uniform distribution has a density function that has the same value at each point. Notice that a uniform density on an infinite continuous domain isn't meaningful, because it could not be scaled to integrate to one.

Let us assume we have a coin which could be from one of two types; the first type of coin is evenly balanced; the other is wildly unbalanced. We flip our coin some number of times, observe the results, and should like to know what type of coin we have. Assume that we flip the coin once. The set of outcomes is

$$D = \{(\texttt{heads}, I), (\texttt{heads}, II), (\texttt{tails}, I), (\texttt{tails}, II)\}$$

An appropriate event space is:

$$\left\{ \begin{array}{cc} \emptyset, & D, \\ \{(\texttt{heads}, I)\}, & \{(\texttt{heads}, II)\}, \\ \{(\texttt{tails}, I)\}, & \{(\texttt{tails}, II)\}, \\ \{(\texttt{heads}, I), (\texttt{heads}, II)\}\,, & \{(\texttt{tails}, I), (\texttt{tails}, II), \}\,, \\ \{(\texttt{tails}, I), (\texttt{heads}, I)\}\,, & \{(\texttt{tails}, II), (\texttt{heads}, II)\}\,, \\ \{(\texttt{heads}, II), (\texttt{tails}, I), (\texttt{tails}, II)\}, & \{(\texttt{heads}, I), (\texttt{tails}, I), (\texttt{tails}, II)\} \\ \{(\texttt{heads}, I), (\texttt{heads}, II), (\texttt{tails}, II)\} & \{(\texttt{heads}, I), (\texttt{heads}, II), (\texttt{tails}, I)\} \end{array} \right\}$$

In this case, assume that we know $P(\text{face}, \text{type})$, for each face and type. Now, for example, the event that the coin shows heads (whatever the type) is represented by the set

$$\{(\texttt{heads}, I), (\texttt{heads}, II)\}$$

We can compute the probability that the coin shows heads (whatever the type) as follows

$$\begin{aligned} P(\{(\texttt{heads}, I), (\texttt{heads}, II)\}) &= P((\texttt{heads}, I) \cup (\texttt{heads}, II)) \\ &= P((\texttt{heads}, I)) + P((\texttt{heads}, II)) \end{aligned}$$

We can compute the probability that the coin is of type I, etc. with similar ease using the same line of reasoning, which applies quite generally.

Example 6.15: *Marginalising out parameters for two different types of coin.*

The **binomial distribution** applies to situations where one has independent identically distributed samples from a distribution with two values. For example, consider drawing n balls from an urn containing equal numbers of black and white balls. Each time a ball is drawn, its colour is recorded and it is replaced, so that the probability of getting a white ball — which we denote p — is the same for each draw. The binomial distribution gives the probability of getting k white balls

$$\left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

The mean of this distribution is np and the variance is $np(1-p)$.

The **Poisson distribution** applies to spatial models that have uniformity properties. Assume that points are placed on the real line randomly in such a way that the expected number of points in an interval is proportional to the length of the interval. The number of points in a unit interval will have a Poisson distribution where

$$P(\{N = x\}) = \frac{\lambda^x e^{-x}}{x!}$$

(where $x = 0, 1, 2 \ldots$ and $\lambda > 0$ is the constant of proportionality). The mean of this distribution is λ and the variance is λ

6.4.1 The Normal Distribution

The probability density function for the **normal distribution** for a single random variable x is

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp - \left\{ \frac{(x-\mu)^2}{2\sigma^2} \right\}$$

The mean of this distribution is μ and the standard deviation is σ. This distribution is widely called a **Gaussian distribution** in the vision community.

The **multivariate normal distribution** for d-dimensional vectors $\boldsymbol{x}$ has probability density function

$$p(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \det(\Sigma)} \exp - \left\{ \frac{(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})}{2} \right\}$$

The mean of this distribution is $\boldsymbol{\mu}$ and the covariance is Σ. Again, this distribution is widely called a **Gaussian distribution** in the vision community.

The normal distribution is extremely important in practice, for several reasons:

- The sum of a large number of random variables is normally distributed, pretty much whatever the distribution of the individual random variables. This fact is known as the **central limit theorem**. It is often cited as a reason to model a collection of random effects with a single normal model.
- Many computations that are prohibitively hard for any other case are easy for the normal distribution.
- In practice, the normal distribution appears to give a fair model of some kinds of noise.
- Many probability density functions have a single peak and then die off; a model for such distributions can be obtained by taking a Taylor series of the log of the density at the peak. The resulting model is a normal distribution (which is often quite a good model).

6.5 Probabilistic Inference

Very often, we have a sequence of observations produced by some process whose mechanics we understand, but which has some underlying parameters that we do not know. The problem is to make useful statements about these parameters. For example, we might observe the intensities in an image, which are produced by the interaction of light and surfaces by principles we understand; what we don't know — and would like to know — are such matters as the shape of the surface, the reflectance of the surface, the intensity of the illuminant, etc. Obtaining some representation of the parameters from the data set is known as **inference**. There is no canonical inference scheme; instead, we need to choose some principle that identifies the most desirable set of parameters.

6.5.1 The Maximum Likelihood Principle

A general inference strategy known as **maximum likelihood estimation**, can be described as

> *Choose the world parameters that maximise the probability of the measurement observed*

In the general case, we are choosing

$$\arg\max P(\text{measurements}|\text{parameters})$$

(where the maximum is only over the world parameters because the measurements are known, and arg max means "the argument that maximises"). In many problems, it is quite easy to specify the measurements that will result from a particular setting of model parameters — this means that $P(\text{measurements}|\text{parameters})$, often referred to as the **likelihood**, is easy to obtain. This can make maximum likelihood estimation attractive.

We return to example 15. Now assume that we know some conditional probabilities. In particular, the unbiased coin has $P(\texttt{heads}|I) = P(\texttt{tails}|I) = 0.5$, and the biased coin has $P(\texttt{tails}|II) = 0.2$ and $P(\texttt{heads}|II) = 0.8$.
We observe a series of flips of a single coin, and wish to know what type of coin we are dealing with. One strategy for choosing the type of coin represented by our evidence is to choose either I or II, depending on whether $P(\text{side}|I) > P(\text{side}|II)$. For example, if we observe four heads and one tail in sequence, then $P(\text{hhhht}|II) = (0.8)^4 0.2 = 0.08192$ and $P(\text{hhhht}|I) = 0.03125$, and we choose type II.

Example 6.16: *Maximum likelihood inference on the type of a coin from its behaviour.*

Maximum likelihood is often an attractive strategy, because it can admit quite simple computation. A classical application of maximum likelihood estimation involves estimating the parameters of a normal distribution from a set of samples of that distribution (example 17).

Assume that we have a set of n samples — the i'th of which is x_i — that are known to be independent and to have been drawn from the same normal distribution. The likelihood of our sample is

$$L(x_1, \ldots x_n; \mu, \sigma) = \prod_i p(x_i; \mu, \sigma) = \prod_i \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

Working with the log of the likelihood will remove the exponential, and not change the position of the maximum. For the log-likelihood, we have

$$Q(x_1, \ldots x_n; \mu, \sigma) = -\sum_i \frac{(x_i - \mu)^2}{2\sigma^2} - n(\frac{1}{2}\log 2 + \frac{1}{2}\log \pi + \log \sigma)$$

and we want the maximum with respect to μ and σ. This must occur when the derivatives are zero, so we have

$$\frac{\partial Q}{\partial \mu} = 2\sum_i \frac{(x_i - \mu)}{2\sigma^2} = 0$$

and a little shuffling of expressions shows that this maximum occurs at

$$\mu = \frac{\sum_i x_i}{n}$$

Similarly

$$\frac{\partial Q}{\partial \sigma} = \frac{\sum_i (x_i - \mu)^2}{\sigma^3} - \frac{n}{\sigma} = 0$$

and this maximum occurs at

$$\sigma = \frac{\sqrt{\sum_i (x_i - \mu)^2}}{n}$$

Note that this estimate of σ is biased, in that its expected value is

$$\sigma(n/(n-1))$$

and it is more usual to use

$$\frac{\sqrt{\sum_i (x_i - \mu)^2}}{(n-1)}$$

as an estimate.

Example 6.17: *Estimating the parameters of a normal distribution from a series of independent samples from that distribution.*

6.5.2 Priors, Posteriors and Bayes' rule

In example 16, our maximum likelihood estimate incorporates no information about $P(I)$ or $P(II)$ — which can be interpreted as how often coins of type I or type II are handed out, or as our subjective degree of belief that we have a coin of type I or of type II before we flipped the coin. This is unfortunate, to say the least; for example, if coins of type II are rare, we would want to see an awful lot of heads before it would make sense to infer that our coin is of this type. Some quite simple algebra suggests a solution.

Recall that $P(A, B) = P(A|B)P(B)$. This simple observation gives rise to an innocuous looking identity for reversing the order in a conditional probability:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

This is widely referred to as **Bayes' theorem** or **Bayes' rule**.

Now the interesting property of Bayes' rule is that it tells us which choice of parameters is most probable, given our model and our prior beliefs. Rewriting Bayes' rule gives

$$P(\text{parameters}|\text{measurements}) = \frac{P(\text{measurements}|\text{parameters})P(\text{parameters})}{P(\text{measurements})}$$

The term $P(\text{parameters})$ is referred to as the **prior** (presumably because it describes our knowledge of the world *before* measurements have been taken). The term $P(\text{parameters}|\text{data})$ is usually referred to as the **posterior** (presumably because it describes the probability of various models *after* measurements have been taken). $P(\text{data})$ can be computed by marginalisation (which requires computing a high dimensional integral, often a nasty business) or for some problems can be ignored. As we shall see in following sections, attempting to use Bayes' rule can result in difficult computations — that integral being one — because posterior distributions often take quite unwieldy forms.

6.5.3 Bayesian Inference

The Bayesian philosophy is that

> all information about the world is captured by the posterior.

The first reason to accept this view is that the posterior is a principled combination of prior information about the world and a model of the process by which measurements are generated — i.e. there is no information missing from the posterior, and the information that is there, is combined in a proper manner. The second reason is that the approach appears to produce very good results. The great difficulty is that computing with posteriors can be very difficult — we will discuss mechanisms for computing with posteriors in section **??**.

For example, we could use the study of physics in the last few chapters to get expressions relating pixel values to the position and intensity of light sources, the reflectance and orientation of surfaces, etc. Similarly, we are likely to have some beliefs about the parameters that have nothing to do with the particular values of the measurements that we observe. We know that albedos are never outside the range $[0, 1]$; we expect that illuminants with extremely high exitance are uncommon; and we expect that no particular surface orientation is more common than any other. This means that we can usually cobble up a reasonable choice of $P(\text{parameters})$. This expression is usually referred to as a **prior**, because it represents beliefs about the state of the world before measurements were made.

MAP Inference

An alternative to maximum likelihood inference is to infer a state of the world that maximises the posterior:

> *Choose the world parameters that maximise the conditional probability of the parameters, conditioned on the measurements taking the observed values*

This approach is known as **maximum *a posteriori*** (or MAP) reasoning.

Assume that we have three flips of the coin, and would like to determine whether it has type I or type II. We know that the mint has 3 machines that produce type I coins and 1 machine that produces type II coins, and there is no reason to believe that these machines run at different rates. We therefore assign $P(I) = 0.75$ and $P(II) = 0.25$. Now we observe three heads, in three consecutive flips. The value of the posterior for type I is:

$$\begin{aligned} P(I|\text{hhh}) &= \frac{P(\text{hhh}|I)P(I)}{P(\text{hhh})} \\ &= \frac{P(\text{h}|I)^3 P(I)}{P(\text{hhh}, I) + P(\text{hhh}, II)} \\ &= \frac{P(\text{h}|I)^3 P(I)}{P(\text{hhh}|I)P(I) + P(\text{hhh}|II)P(II)} \\ &= \frac{0.5^3 0.75}{0.5^3 0.75 + 0.8^3 0.25} \\ &= 0.422773 \end{aligned}$$

By a similar argument, the value of the posterior for type II is 0.577227. An MAP inference procedure would conclude the coin is of type II.

Example 6.18: *Determining the type of a coin using MAP inference.*

The denominator in the expression for the posterior can be quite difficult to compute, because it requires a sum over what is potentially a very large number of elements (imagine what would happen if there were many different types of coin). However, knowing this term is not crucial if we wish to isolate the element with the maximum value of the posterior, because it is a constant. Of course, if there are a very large number of events in the discrete space, finding the world parameters that maximise the posterior can be quite tricky.

The Posterior as an Inference

Assume we have a coin which comes from a mint which has a continuous control parameter, λ, which lies in the range $[0, 1]$. This parameter gives the probability that the coin comes up heads, so $P(\texttt{heads}|\lambda) = \lambda$. We know no reason to prefer any one value of λ to any other, so as a prior probability distribution for λ we use the uniform distribution so $p(\lambda) = 1$.
Assume we flip the coin twice, and observe heads twice; what do we know about λ? All our knowledge is captured by the posterior, which is

$$\frac{P(\lambda \in [x, x + dx]|\texttt{hh})}{dx}$$

we shall write this expression as $p(\lambda|\texttt{hh})$. We have

$$\begin{aligned} p(\lambda|\texttt{hh}) &= \frac{p(\texttt{hh}|\lambda)p(\lambda)}{p(\texttt{hh})} \\ &= \frac{p(\texttt{hh}|\lambda)p(\lambda)}{\int_0^1 p(\texttt{hh}|\lambda)p(\lambda)d\lambda} \\ &= \frac{\lambda^2 p(\lambda)}{\int_0^1 p(\texttt{hh}|\lambda)p(\lambda)d\lambda} \\ &= 3\lambda^2 \end{aligned}$$

It is fairly easy to see that if we flip the coin n times, and observe k heads and $n - k$ tails, we have

$$p(\lambda|k \text{ heads and } n - k \text{ tails}) \propto c\lambda^k(1 - \lambda)^{n-k}$$

Example 6.19: *Determining the probability a coin comes up heads from the outcome of a sequence of flips.*

We have argued that choosing parameters that maximise the posterior is a useful inference mechanism. But, as figure 6.2 indicates, the posterior is good for other uses as well. This figure plots the posterior distribution on the probability that a coin comes up heads, given the result of some number of flips. In the figure, the posterior distributions indicate not only the single "best" value for the probability that a coin comes up heads, but also the extent of the uncertainty in that value. For example, inferring a value of this probability after two coin flips leads to a value that is not particularly reliable — the posterior is a rather flat function, and there are many different values of the probability with about the same value of the posterior. Possessing this information allows us to compare this evidence with other sources of evidence about the coin.

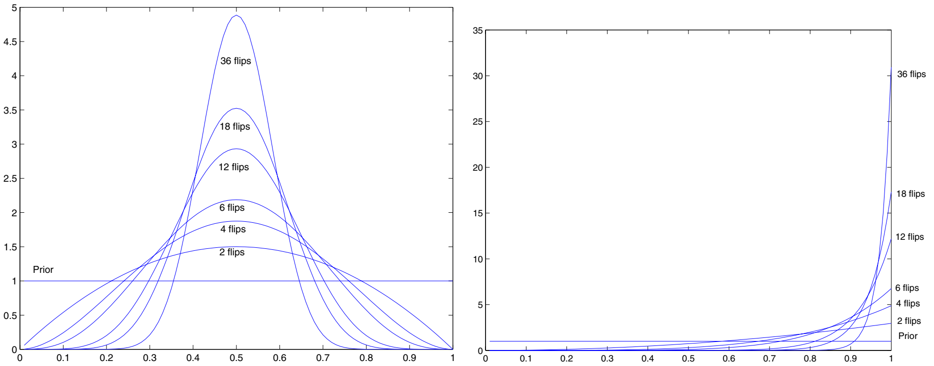


Figure 6.2. On the left, the value of the posterior density for the probability that a coin will come up heads, given an equal number of heads and tails are observed. This posterior is shown for different numbers of observations. With no evidence, the posterior is the prior; but as the quantity of evidence builds up, the posterior becomes strongly peaked — this is because one is very unlikely to observe a long sequence of coin flips where the frequency of heads is very different from the probability of obtaining a head. On the right, a similar plot, but now for the case where every flip comes up heads. As the number of flips builds up, the posterior starts to become strongly peaked near one. This overwhelming of the prior by evidence is a common phenomenon in Bayesian inference.

Bayesian inference is a framework within which it is particularly easy to combine various types of evidence, both discrete and continuous. It is often quite easy to set up the sums.

Bayesian Model Selection

The crucial virtue of Bayesian inference is the accounting for uncertainty shown in examples 20 and 21. We have been able to account for an occasionally untruthful informant and a random measurement; when there was relatively little contradictory evidence from the coin's behaviour, our process placed substantial weight on the informant's testimony, but when the coin disagreed, the informant was discounted. This behaviour is highly attractive, because we are able to combine uncertain sources of information with confidence.

Example 22 shows how to tell whether the informant of examples 20 and 21 is

We use the basic setup of example 19. Assume you have a contact at the coin factory, who will provide a single estimate of λ. Your contact has poor discrimination, and can tell you only whether λ is `low`, `medium` or `high` (i.e in the range $[0, 1/3]$, $(1/3, 2/3)$ or $[2/3, 1]$). You expect that a quarter of the time your contact, not being habitually truthful, will simply guess rather than checking how the coin machine is set. What do you know about λ after a single coin flip, which comes up heads, if your contact says `high`? We need

$$\begin{aligned} p(\lambda|\texttt{high}, \texttt{heads}) &= \frac{p(\texttt{high}, \texttt{heads}|\lambda)p(\lambda)}{p(\texttt{high}, \texttt{heads})} \\ &\propto p(\texttt{high}, \texttt{heads}|\lambda)p(\lambda) \end{aligned}$$

The interesting modelling problem is in $p(\texttt{high}, \texttt{heads}|\lambda)$. This is

$$\begin{aligned} p(\texttt{high}, \texttt{heads}|\lambda) &= p(\texttt{high}, \texttt{heads}|\lambda, \text{truth} = 1)p(\text{truth} = 1) \\ &\quad + p(\texttt{high}, \texttt{heads}|\lambda, \text{truth} = 0)p(\text{truth} = 0) \\ &= p(\texttt{high}, \texttt{heads}|\lambda, \text{truth} = 1)p(\text{truth} = 1) \\ &\quad + p(\texttt{heads}|\lambda, \text{truth} = 0)p(\texttt{high}|\lambda, \text{truth} = 0)p(\text{truth} = 0) \end{aligned}$$

Now from the details of the problem

$$\begin{aligned} p(\text{truth} = 1) &= 0.75 \\ p(\text{truth} = 0) &= 0.25 \\ p(\texttt{heads}|\lambda, \text{truth} = 0) &= \lambda \\ p(\texttt{high}|\lambda, \text{truth} = 0) &= \frac{1}{3} \end{aligned}$$

and the term to worry about is $p(\texttt{high}, \texttt{heads}|\lambda, \text{truth} = 1)$. This term reflects the behaviour of the coin and the informant when the informant is telling the truth; in particular, this term must be zero for $\lambda \in [0, 2/3)$, because in this case λ is not `high`, so we never see a truthful report of `high` with λ in this range. For λ in the `high` range, this term must be λ, because now it is the probability of getting a head with a single flip. Performing the computation, we obtain the posterior graphed in figure 6.3.

Example 6.20: *Determining the type of a coin from a sequence of flips, incorporating information from an occasionally untruthful informant.*

telling the truth or not, given the observations. A useful way to think about this example is to regard it as comparing two *models* (as opposed to the value of a binary parameter within one model). One model has a lying informant, and the other has a truthful informant. The posteriors computed in this example compare how well

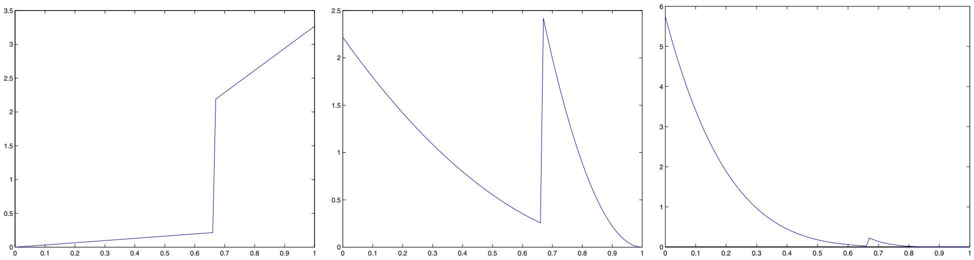


Figure 6.3. On the left, the posterior probability density for the probability a coin comes up heads, given a single flip that shows a head and a somewhat untruthful informant who says `high`, as in example 20. In the center, a posterior probability density for the same problem, but now assuming that we have seen two tails and the informant says `high` (a sketch of the formulation appears in example 21). On the right, a posterior probability density for the case when the coin shows five tails and the informant says `high`. As the number of tails builds up, the weight of the posterior in the `high` region goes down, strongly suggesting the informant is lying.

Now consider what happens in example 20 if the contact says high and we see two tails. We need

$$\begin{aligned} p(\lambda|\texttt{high}, \texttt{tt}) &= \frac{p(\texttt{high}, \texttt{tt}|\lambda)p(\lambda)}{p(\texttt{high}, \texttt{tt})} \\ &\propto p(\texttt{high}, \texttt{tt}|\lambda)p(\lambda) \end{aligned}$$

Now $p(\texttt{high}, \texttt{tt}|\lambda)$ is

$$\begin{aligned} p(\texttt{high}, \texttt{tt}|\lambda) &= p(\texttt{high}, \texttt{tt}|\lambda, \text{truth} = 1)P(\text{truth} = 1) \\ &\quad + p(\texttt{high}, \texttt{tt}|\lambda, \text{truth} = 0)P(\text{truth} = 0) \\ &= p(\texttt{high}, \texttt{tt}|\lambda, \text{truth} = 1)P(\text{truth} = 1) \\ &\quad + p(\texttt{tt}|\lambda, \text{truth} = 0)p(\texttt{high}|\lambda, \text{truth} = 0)P(\text{truth} = 0) \end{aligned}$$

Now $p(\texttt{tt}|\lambda, \text{truth} = 0) = (1 - \lambda)^2$ and the interesting term is $p(\texttt{high}, \texttt{tt}|\lambda, \text{truth} = 1)$. Again, this term reflects the behaviour of the coin and the informant when the informant is telling the truth; in particular, this term must be zero for $\lambda \in [0, 2/3)$, because in this case λ is not high. For λ in the high range, this term must be $(1 - \lambda)^2$, because now it is the probability of getting two tails with two flips. Performing the computation, we obtain the posterior graphed in figure 6.3.

Example 6.21: *Determining the type of a coin from a sequence of flips, incorporating information from an occasionally untruthful informant — II.*

We now need to know whether our informant lied to us. Assume we see a single head and an informant saying `high`, again. The relevant posterior is:

$$\begin{aligned}
P(\text{truth}=0|\texttt{head},\texttt{high}) &= \frac{P(\texttt{head},\texttt{high}|\text{truth}=0)P(\text{truth}=0)}{P(\texttt{head},\texttt{high})} \\
&= \frac{\int P(\lambda,\texttt{head},\texttt{high}|\text{truth}=0)P(\text{truth}=0)d\lambda}{P(\texttt{head},\texttt{high})} \\
&= \frac{\int P(\texttt{head},\texttt{high}|\lambda,\text{truth}=0)P(\lambda)P(\text{truth}=0)d\lambda}{P(\texttt{head},\texttt{high})} \\
&= \frac{1}{1+\frac{\int P(\texttt{head},\texttt{high}|\lambda,\text{truth}=1)P(\lambda)d\lambda P(\text{truth}=1)}{\int P(\texttt{head},\texttt{high}|\lambda,\text{truth}=0)P(\lambda)d\lambda P(\text{truth}=0)}}
\end{aligned}$$

Example 6.22: *Is the informant lying?*

different models explain a given data set, given a prior on the *models*. This is a very general problem — usually called **model selection** — with a wide variety of applications in vision:

- **Recognition:** Assume we have a region in an image, and an hypothesis that an object might be present in that region at a particular position and orientation (the hypothesis will have been obtained using methods from section **??**, which aren't immediately relevant). Is there an object there or not? A principled answer involves computing the posterior over two models — that the data was obtained from noise, or from the presence of an object.

- **Are these the same?** Assume we have a set of pictures of surfaces we want to compare. For example, we might want to know if they are the same colour, which would be difficult to answer directly if we didn't know the illuminant. A principled answer involves computing the posterior over two models — that the data was obtained from one surface, or from two (or more).

- **What camera was used?** Assume we have a sequence of pictures of a world. With a certain amount of work, it is usually possible to infer a great deal of information about the shape of the objects from such a sequence (section **??**). The algorithms involved differ quite sharply, depending on the camera model adopted (i.e. perspective, orthographic, etc.). Furthermore, adopting the wrong camera model tends to lead to poor inferences. Determining the right camera model to use is quite clearly a model selection problem.

- **How many segments are there?** We would like to break an image into coherent components, each of which is generated by a probabilistic model. How many components should there be? (section 17.3).

The solution is so absurdly simple *in principle* (in practice, the computations can be quite nasty) that it is easy to expect something more complex, and miss it. We will write out Bayes' rule specialised to this case to avoid this:

$$\begin{aligned} P(\text{model}|\text{data}) &= \frac{P(\text{data}|\text{model})}{P(\text{data})} \\ &= \frac{\int P(\text{data}|\text{model}, \text{parameters})P(\text{parameters})d\{\text{parameters}\}}{P(\text{data})} \\ &\propto \int P(\text{data}|\text{model}, \text{parameters})P(\text{parameters})d\{\text{parameters}\} \end{aligned}$$

which is exactly the form used in the example. Notice that we are engaging in Bayesian inference here, too, and so can report the MAP solution or report the whole posterior. The latter can be quite helpful when it is difficult to distinguish between models. For example, in the case of the dodgy informant, if $P(\text{truth}{=}0|\text{data}) = 0.5001$, it may be undesirable to conclude the informant is lying — or at least, to take drastic action based on this conclusion. The integral is potentially rather nasty, which means that the method can be quite difficult to use in practice. We will discuss methods for computing the integral in section **??**; useful references include [].

6.5.4 Open Issues

In the rest of the book, we will have regular encounters with practical aspects of the Bayesian philosphy. Firstly, although the posterior encapsulates all information available about the world, we very often need to make discrete decisions — should we shoot it or not? Typically, this decision making process requires some accounting for the cost of false positives and false negatives.

Secondly, how do we build models? There are three basic sources of likelihood functions and priors:

- **Judicious design:** it is possible to come up with models that are too hard to handle computationally. Generally, models on very high-dimensional domains are difficult to deal with, particularly if there is a great deal of interdependence between variables. There is a family of models — commonly known as **graphical models** — for which quite good inference algorithms are known. The underlying principle of this approach is to exploit simplifications due to independence and conditional independence. We describe this approach in chapter **??**, section **??** and in chapter **??**, section **??**, in the context of relevant examples.

- **Physics:** particularly in low-level vision problems, likelihood models follow quite simply from physics. It is hard to give a set of design rules for this strategy. Instead, we illustrate the approach with an example, in section **??**.

- **Learning:** as section **??** suggested, a poor choice of model results in poor performance, and a good choice of model results in good performance. We can use this observation to tune the structure of models if we have a sufficient set of data. We describe aspects of this strategy in chapter **??**.

Finally, the examples above suggest that posteriors can have a nasty functional form. This intuition is correct, and there is a body of technique that can help handle ugly posteriors which we explore in section **??**.

6.6 Discussion

Probabilistic inference techniques lie at the core of any solution to serious vision problems. The great difficulty, in our opinion, is arriving at a model that is both sufficiently accurate and sufficiently compact to allow useful inference. This isn't at all easy. A naive Bayesian view of vision — write out a posterior using the physics of illumination and reflection, guess some reasonable priors, and then study the posterior — very quickly falls apart. In terms of what representation should this posterior be written? and how can we extract information from the posterior? These questions are exciting research topics.

The examples in this chapter are all pretty simple, so as to expose the line of reasoning required. We do some hard examples in chapter **??**. Building and handling complex examples is still very much a research topic; however, probabilistic reasoning of one form or another is now pervasive in vision, which is why it's worth studying.

Exercises

1. The event structure of section 6.1 did not explicitly include unions. Why does the text say that unions are here?
2. In example **??**, if $P(\text{heads}) = p$, what is $P(\text{tails})$?
3. In example 10 show that if $P(\text{hh}) = p^2$ then $P(\{\text{ht}, \text{th}\}) = 2p(1-p)$ and $P(\text{tt}) = (1-p)^2$.
4. In example 10 it says that

$$P(k \text{ heads and } n-k \text{ tails in } n \text{ flips}) = \left(\begin{array}{c} k \\ n \end{array} \right) p^k (1-p)^{n-k}$$

   Show that this is true.

5. A careless study of example 10 often results in quite muddled reasoning, of the following form: I have bet on heads successfully ten times, therefore I should bet on tails next. Explain why this muddled reasoning — which has its own name, **the gambler's fallacy** in some circles, **anti-chance** in others — is muddled.
6. Confirm the count of parameters in example 8.

7. In example 19, what is c?

8. As in example 16, you are given a coin of either type I or type II; you do not know the type. You flip the coin n times, and observe k heads. You will infer the type of the coin using maximum likelihood estimation. for what values of k do you decide the coin is of type I?

9. Compute $P(\text{truth}|\text{high}, \text{coin behaviour})$ for each of the three cases of example 21. You'll have to estimate an integral numerically.

10. In example 22, what is the numerical value of the probability that the informant is lying, given that the informant said `high` and the coin shows a single `tail`? What is the numerical value of the probability that the informant is lying, given that the informant said `high` and the coin shows seven `tail`s in eight flips?

11. The random variable $\boldsymbol{x} = (x_1, x_2, \ldots x_n)^T$ has a normal distribution. Show that the random variable $\hat{\boldsymbol{x}} = (x_2, \ldots, x_n)^T$ has a normal distribution (which is obtained by marginalizing the density). A good way to think about this problem is to consider the mean and covariance of $\hat{\boldsymbol{x}}$, and reason about the behaviour of the integral; a bad way is to storm ahead and try and do the integral.

12. The random variable $\boldsymbol{p}$ has a normal distribution. Furthermore, there are symmetric matrices $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ and vectors $\boldsymbol{D}$ and $\boldsymbol{E}$ such that $P(\boldsymbol{d}|\boldsymbol{p})$ has the form

$$-\log P(\boldsymbol{d}|\boldsymbol{p}) = \boldsymbol{p}^T \mathcal{A} \boldsymbol{p} + \boldsymbol{p}^T \mathcal{B} \boldsymbol{d} + \boldsymbol{d}^T \mathcal{C} \boldsymbol{d} + \boldsymbol{p}^T \boldsymbol{D} + \boldsymbol{d}^T \boldsymbol{E} + C$$

(C is the log of the normalisation constant). Show that $P(\boldsymbol{p}|\boldsymbol{d})$ is a normal distribution for any value of $\boldsymbol{d}$. This has the great advantage that inference is relatively easy.

13. x is a random variable with a continuous cumulative distribution function $F(x)$. Show that $u = F(x)$ is a random variable with a uniform density on the range $[0, 1]$. Now use this fact to show that $w = F^{-1}(u)$ is a random variable with cumulative distribution function F.

Part III

EARLY VISION: ONE IMAGE

Chapter 7

LINEAR FILTERS

Pictures of zebras and of dalmatians have black and white pixels, and in about the same number, too. The differences between the two have to do with the characteristic appearance of small groups of pixels, rather than individual pixel values. In this chapter, we introduce methods for obtaining descriptions of the appearance of a small group of pixels.

Our main strategy will be to use weighted sums of pixel values, using different patterns of weights to find different image patterns. This process, despite its simplicity, is extremely useful. It allows us to smooth noise in images, and to find edges and other image patterns. We discuss noise in some detail. We also describe some useful non-linear functions of image neighbourhoods.

7.1 Linear Filters and Convolution

Many important effects can be modelled with a quite simple model. Construct a new array, the same size as the image. Fill each location of this new array with a weighted sum of the pixel values from the locations surrounding the corresponding location in the image, *using the same set of weights each time.* Different sets of weights could be used to represent different processes — for example, we could use a set of weights that was large at the center and fell off sharply as the distance from the center increased to model the kind of smoothing that occurs in a defocussed lens system. The result of this procedure is **shift-invariant** — meaning that the value of the output depends on the pattern in an image neighbourhood, rather than the position of the neighbourhood — and **linear** — meaning that the output for the sum of two images is the same as the sum of the outputs obtained for the images separately. The procedure itself is known as **linear filtering**.

7.1.1 Convolution

We introduce some notation at this point. The pattern of weights used for a linear filter is usually referred to as the **kernel** of the filter. The process of applying the filter is usually referred to as **convolution**. There is a catch: for reasons that will appear later (section 7.2.1), it is convenient to write the process in a non-obvious way.

In particular, given a filter kernel $\mathcal{H}$, the convolution of the kernel with image $\mathcal{F}$ is an image $\mathcal{R}$. The i, j'th component of $\mathcal{R}$ are given by:

$$R_{ij} = \sum_{u,v} H_{i-u,j-v} R_{u,v}$$

We carefully avoid inserting the range of the sum; in effect, we assume that the sum is over a large enough range of u and v that all non-zero values are taken into account. We will use this convention — which is common — regularly in what follows.

7.1.2 Example: Smoothing by Averaging

Images typically have the property that the value of a pixel is usually similar to that of its neighbour. Assume that the image is affected by noise of a form where we can reasonably expect that this property is preserved. For example, there might be occasional dead pixels; or small random numbers with zero mean might have been added to the pixel values. It is natural to attempt to reduce the effects of this noise by replacing each pixel with a weighted average of its neighbours, a process often referred to as **smoothing** or **blurring**.

At first guess, we could model the blurring process as replacing the value of a function at each point with an unweighted (or uniform) average taken over a fixed region. For example, we could average all pixels within a $2k + 1 \times 2k + 1$ block of the pixel of interest. For an input image $\mathcal{F}$, this gives an output

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}$$

This is the same as convolution with a kernel that is a $2k+1 \times 2k+1$ block of ones, multiplied by a constant — you can establish this point by close attention to the range of the sum.

This process is a poor model of blurring — its output does not look like that of a defocussed camera (figure 7.1. The reason is clear. Assume that we have an image in which every point but the center point was zero, and the center point was one. If we blur this image by forming an unweighted average at each point, the result will look like a small bright box — but this is not what defocussed cameras do. We want a blurring process that takes a very small bright dot to a circularly symmetric region of blur, brighter at the center than at the edges and fading slowly to darkness. As figure 7.1 suggests, a set of weights of this form produces a much more convincing defocus model.

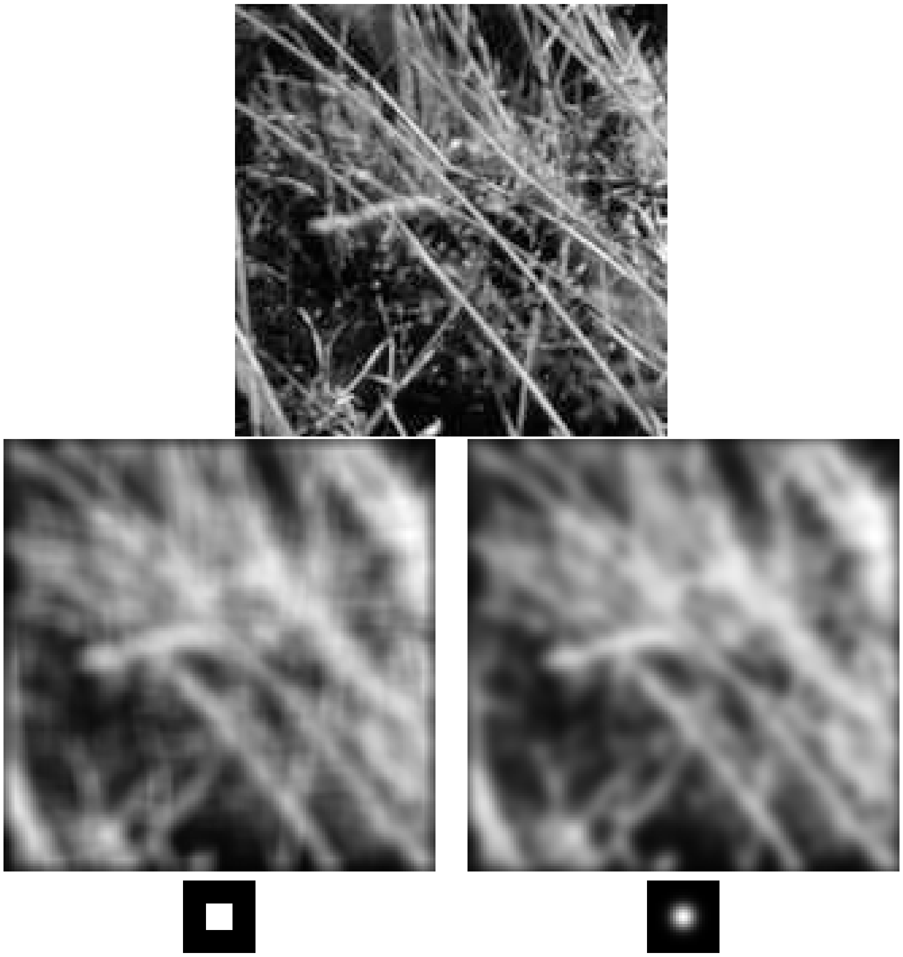


Figure 7.1. Although a uniform local average may seem to give a good blurring model, it generates effects that are not usually seen in defocussing a lens. The images above compare the effects of a uniform local average with weighted average. The image at the top shows a view of grass. On the left in the second row, the result of blurring this image using a uniform local model and on the right, the result of blurring this image using a set of Gaussian weights. The degree of blurring in each case is about the same, but the uniform average produces a set of narrow vertical and horizontal bars — an effect often known as **ringing**. The bottom row shows the weights used to blur the image, themselves rendered as an image; bright points represent large values and dark points represent small values (in this example the smallest values are zero).

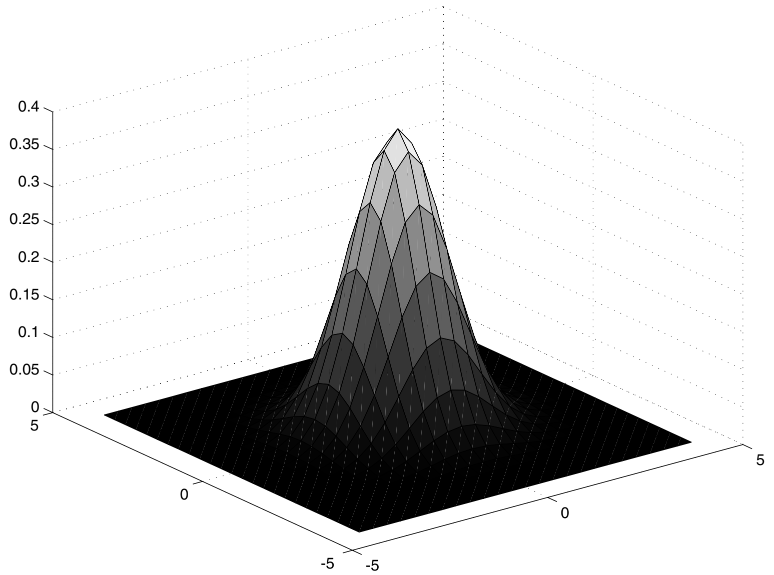


Figure 7.2. The symmetric Gaussian kernel in 2D. This view shows a kernel scaled so that its sum is equal to one; this scaling is quite often omitted. The kernel shown has $\sigma = 1$. Convolution with this kernel forms a weighted average which stresses the point at the center of the convolution window, and incorporates little contribution from those at the boundary. Notice how the Gaussian is qualitatively similar to our description of the point spread function of image blur; it is circularly symmetric, has strongest response in the center, and dies away near the boundaries.

7.1.3 Example: Smoothing with a Gaussian

A good formal model for this fuzzy blob is the **symmetric Gaussian kernel**

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

illustrated in figure 7.2. σ is referred to as the standard deviation of the Gaussian (or its "sigma"!); the units are inter-pixel spaces, usually referred to as pixels. The constant term makes the integral over the whole plane equal to one and is often ignored in smoothing applications. The name comes from the fact that this kernel has the form of the probability density for a 2D Gaussian random variable with a particular covariance.

This smoothing kernel forms a weighted average, that weights pixels at its center much more strongly than at its boundaries. One can justify this approach qualitatively: smoothing suppresses noise by enforcing the requirement that pixels should look like their neighbours; and by down-weighting distant neighbours in the average, we can ensure that the requirement that a pixel look like its neighbours is less strongly imposed for distant neighbours. A qualitative analysis gives:

- if standard deviation of the Gaussian is very small — say smaller than one pixel — the smoothing will have very little effect, because the weights for all pixels off the center will be very small;
- for a larger standard deviation, the neighbouring pixels will have larger weights in the weighted average, which means in turn that the average will be strongly biased toward a consensus of the neighbours — this will be a good estimate of a pixel's value, and the noise will largely disappear, at the cost of some blurring;
- finally, a kernel that has very large standard deviation will cause much of the image detail to disappear along with the noise.

Figure 7.3 illustrates these phenomena. You should notice that Gaussian smoothing can be effective at suppressing noise.

In applications, a discrete smoothing kernel is obtained by constructing a $2k + 1 \times 2k + 1$ array whose i, j'th value is

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{((i-k-1)^2 + (j-k-1)^2)}{2\sigma^2}\right)$$

Notice that some care must be exercised with σ; if σ is too small, then only one element of the array will have a non-zero value. If σ is large, then k must be large, too, otherwise we will be ignoring contributions from pixels that should contribute with substantial weight.

7.2 Shift invariant linear systems

Most imaging systems have, to a good approximation, three significant properties:

- **Superposition:** we expect that

  $$R(f + g) = R(f) + R(g)$$

  that is, the response to the sum of stimuli is the sum of the individual responses.
- **Scaling:** the response to a zero input is zero. Taken with superposition, we have that the response to a scaled stimulus is a scaled version of the response to the original stimulus, i.e.

  $$R(kf) = kR(f)$$

  A device that exihibits superposition and scaling is **linear**.

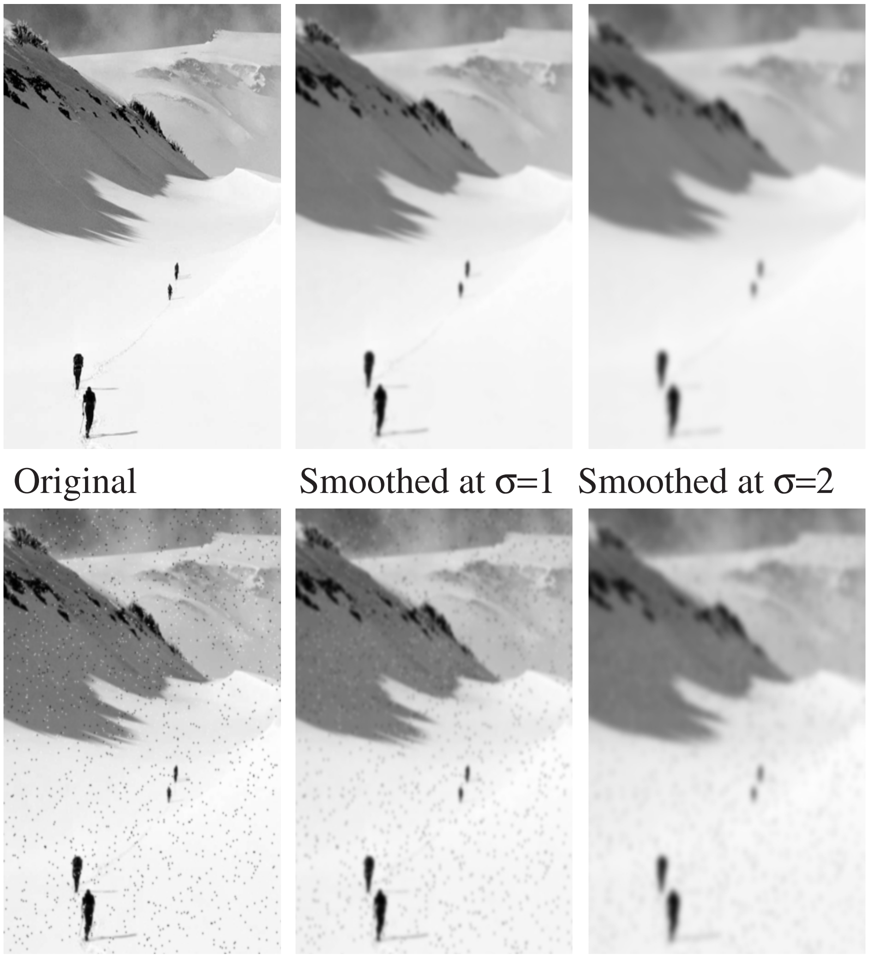


Figure 7.3. In salt-and-pepper noise, we choose pixels uniformly at random, and uniformly at random make them either black or white. Gaussian smoothing is particularly effective at suppressing the effects of salt-and-pepper noise. The top row shows an image, and versions smoothed by a symmetric Gaussian with σ two pixels and four pixels. The images in the second row are obtained by corrupting the images in the top row by this noise model and then smoothing the result. Notice that, as the smoothing increases, detail is lost, but the effects of the noise diminish, too — the smoothed versions of the noisy images look very much like the smoothed version of the noise-free images.

- **Shift invariance:** in a **shift invariant** system, the response to a translated stimulus is just a translation of the response to the stimulus. This means that, for example, if a view of a small light aimed at the center of the camera is a small bright blob, then if the light is moved to the periphery, we should see the same small bright blob, only translated.

A device that is linear and shift invariant is known as a **shift invariant linear system**, or often just as a **system**.

The response of a shift invariant linear system to a stimulus is obtained by convolution. We will demonstrate this first for systems that take discrete inputs — say vectors or arrays — and produce discrete outputs. We then use this to describe the behaviour of systems which operate on continuous functions of the line or the plane, and from this analysis obtain some useful facts about convolution.

7.2.1 Discrete Convolution

In the 1D case, we have a shift invariant linear system that takes a vector and responds with a vector. This case is the easiest to handle, because there are fewer indices to look after. The 2D case — a system that takes an array, and responds with an array — follows easily. In each case, we assume that the input and output are infinite dimensional. This allows us to ignore some minor issues that arise at the boundaries of the input — we'll deal with these later (section 7.2.3).

Discrete Convolution in One Dimension

We have an input vector $\boldsymbol{f}$. For convenience, we will assume that the vector is infinite, and its elements are indexed by the integers (i.e. there is a -1'th element, etc.). The i'th component of this vector is f_i. Now $\boldsymbol{f}$ is a weighted sum of basis elements. A convenient basis is a set of elements that have a one in one component, and zeros elsewhere. We write

$$\boldsymbol{e}_0 = \ldots 0, 0, 0, 1, 0, 0, 0, \ldots$$

(this is a data vector that has a 1 in the zero'th place, and zeros elsewhere). Define a shift operation, where $\texttt{Shift}(\boldsymbol{f}, i)$ is a vector whose j'th component is the $j-i$'th component of $\boldsymbol{f}$. For example, $\texttt{Shift}(\boldsymbol{e}_0, 1)$ has a zero in the first component. Now we can write

$$\boldsymbol{f} = \sum_i f_i \texttt{Shift}(\boldsymbol{e}_0, i)$$

We write the response of our system to a vector $\boldsymbol{f}$ as

$$R(\boldsymbol{f})$$

Now because the system is shift invariant, we have that

$$R(\texttt{Shift}(\boldsymbol{f}, k)) = \texttt{Shift}(R(\boldsymbol{f}), k)$$

Furthermore, because it is linear, we have that

$$R(k\boldsymbol{f}) = kR(\boldsymbol{f})$$

This means that

$$\begin{aligned} R(\boldsymbol{f}) &= R(\sum_i f_i \texttt{Shift}(\boldsymbol{e}_0, i)) \\ &= \sum_i R(f_i \texttt{Shift}(\boldsymbol{e}_0, i)) \\ &= \sum_i f_i R(\texttt{Shift}(\boldsymbol{e}_0, i)) \\ &= \sum_i f_i \texttt{Shift}(R(\boldsymbol{e}_0), i)) \end{aligned}$$

This means that to obtain the system's response to any data vector, we need to know only its response to $\boldsymbol{e}_0$. This is usually called the system's **impulse response**. Assume that the impulse response can be written as $\boldsymbol{g}$. We have

$$R(\boldsymbol{f}) = \sum_i h_i \texttt{Shift}(\boldsymbol{g}, j) = \boldsymbol{g} * \boldsymbol{h}$$

This defines an operation — the 1D, discrete version of convolution — which we write with a $*$.

This is all very well, but does not give us a particularly easy expression for the output. If we consider the j'th element of $R(\boldsymbol{f})$, which we write as R_i, we must have:

$$R_j = \sum_i g_{j-i} f_i$$

which conforms to (and explains the origin of) the form we used in section 7.1.1.

Discrete Convolution in Two Dimensions

We now use an array of values, and write the i, j'th element of the array $\mathcal{D}$ is D_{ij}. The appropriate analogy to an impulse response is the response to a stimulus that looks like:

$$\mathcal{E}_{00} = \begin{array}{ccccc} \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & 0 & 0 & 0 & \ldots \\ \ldots & 0 & 1 & 0 & \ldots \\ \ldots & 0 & 0 & 0 & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{array}$$

If $\mathcal{G}$ is the response of the system to this stimulus, the same considerations as for 1D convolution yield a response to a stimulus $\mathcal{F}$ that is:

$$R_{ij} = \sum_{u,v} G_{i-u,j-v} F_{uv}$$

which we write as

$$\mathcal{R} = \mathcal{G} * * \mathcal{H}$$

7.2.2 Continuous Convolution

There are shift invariant linear systems that produce a continuous response to a continuous input; for example, a camera lens takes a set of radiances and produces another set, and many lenses are approximately shift invariant. A brief study of these systems will allow us to study the information that is lost by approximating a continuous function — the incoming radiance values across an image plane — by a discrete function — the value at each pixel.

The natural description is in terms of the system's response to a rather unnatural function — the δ-function, which is not a function in formal terms. We will do the derivation first in one dimension, to make the notation easier.

Convolution in One Dimension

We will obtain an expression for the response of a continuous shift invariant linear system from our expression for a discrete system. We can take a discrete input, and replace each value with a box, straddling the value — this gives a continuous input function. We will then make the boxes narrower, and consider what happens in the limit.

Our system takes a function of one dimension and returns a function of one dimension. Again, we write the response of the system to some input $f(x)$ as $R(f)$; when we need to emphasize that f is a function, we write $R(f(x))$. The response is also a *function*; occasionally, when we need to emphasize this fact, we will write $R(f)(u)$. We can express the linearity property in this notation by writing

$$R(kf) = kR(f)$$

(for k some constant) and the shift invariance property by introducing a `Shift` operator, which takes functions to functions:

$$\texttt{Shift}(f, c) = f(u - c)$$

With this `Shift` operator, we can write the shift invariance property as:

$$R(\texttt{Shift}(f, c)) = \texttt{Shift}(R(f), c)$$

We define the *box* function as:

$$box_\epsilon(x) = \begin{cases} 0 & abs(x) > \frac{\epsilon}{2} \\ 1 & abs(x) < \frac{\epsilon}{2} \end{cases}$$

The value of $box_\epsilon(\epsilon/2)$ does not matter for our purposes. The input function is $f(x)$. We construct an even grid of points x_i, where $x_{i+1} - x_i = \epsilon$. We now construct a vector $\boldsymbol{f}$, whose i'th component (written f_i) is $f(x_i)$. This vector can be used to represent the function.

We obtain an approximate representation of f by $\sum_i f_i \texttt{Shift}(box_\epsilon, x_i)$ We apply this input to a shift-invariant linear system; the response is a weighted sum of shifted responses to box functions as in figure **??**. This means that

$$\begin{aligned} R(\sum_i f_i \texttt{Shift}(box_\epsilon, x_i)) &= \sum_i R(f_i \texttt{Shift}(box_\epsilon, x_i)) \\ &= \sum_i f_i R(\texttt{Shift}(box_\epsilon, x_i)) \\ &= \sum_i f_i \texttt{Shift}(R(\frac{box_\epsilon}{\epsilon}\epsilon), x_i) \\ &= \sum_i f_i \texttt{Shift}(R(\frac{box_\epsilon}{\epsilon}), x_i)\epsilon \end{aligned}$$

(so far, everything has followed our derivation for discrete functions). We now have something that looks like an approximate integral, if $\epsilon \to 0$.

We introduce a new device, called a δ-function, to deal with the term box_ϵ/ϵ. Define

$$d_\epsilon(x) = \frac{box_\epsilon(x)}{\epsilon}$$

The δ-function is:

$$\delta(x) = \lim_{\epsilon \to 0} d_\epsilon(x)$$

We don't attempt to evaluate this limit, so we need not discuss the value of $\delta(0)$. One interesting feature of this function is that for practical shift-invariant linear systems the response of the system to a δ-function exists and has **compact support** (i.e. is zero except on a finite number of intervals of finite length). For example, a good model of a δ-function in 2D is an extremely small, extremely bright light. If we make the light smaller and brighter, while ensuring the total energy is constant, we expect to see a small but finite spot due to the defocus of the lens. The δ-function is the natural analogue for $\boldsymbol{e}_0$ in the continuous case.

This means that the expression for the response of the system,

$$\sum_i f_i \texttt{Shift}(R(\frac{box_\epsilon}{\epsilon}), x_i)\epsilon$$

will turn into an integral as ϵ limits to zero. We obtain

$$\begin{aligned} R(f) &= \int \{R(\delta)(u - x')\} f(x')dx' \\ &= \int g(u - x')f(x')dx' \end{aligned}$$

where we have written $R(\delta)$ — which is usually called the **impulse response** of the system — as g and have omitted the limits of the integral. These integral could be from $-\infty$ to ∞, but more stringent limits could apply if g and h have compact support. This operation is called **convolution** (again), and we write the expression above as

$$R(f) = (g * f)$$

Convolution is **symmetric**, meaning

$$(g * h)(x) = (h * g)(x)$$

Convolution is *associative*, meaning that

$$(f * (g * h)) = ((f * g) * h)$$

This latter property means that we can find a single shift-invariant linear system that behaves like the composition of two different systems. This will come in useful when we discuss sampling.

Convolution in Two Dimensions

The derivation of convolution in two dimensions requires more notation — a box function is now given by $box_\epsilon(x, y) = box_\epsilon(x) box_\epsilon(y)$; we now have

$$d_\epsilon(x, y) = \frac{box_\epsilon(x, y)}{\epsilon^2}$$

The δ-function is the limit of $d_\epsilon(x, y)$ function as $\epsilon \to 0$; and there are more terms in the sum. All this activity will result in the expression:

$$\begin{aligned} R(h)(x, y) &= \int \int g(x - x', y - y') h(x', y') dx dy \\ &= (g * * h)(x, y) \end{aligned}$$

Where we have used two $*$'s to indicate a two dimensional convolution. Convolution *in 2D* is **symmetric** — that is $(g * * h) = (h * * g)$ — and associative.

A natural model for the impulse response of two dimensional system is to think of the pattern seen in a camera viewing a very small, distant light source (which subtends a very small viewing angle). In practical lenses, this view results in some form of fuzzy blob, justifying the name **point spread function** which is often used for the impulse response of a 2D system. The point spread function of a linear system is often known as its **kernel**.

7.2.3 Edge Effects in Discrete Convolutions

In practical systems we cannot have infinite arrays of data. This means that, when we compute the convolution, we need to contend with the edges of the image; at the edges, there are pixel locations where computing the value of the convolved image requires image values that don't exist. There are a variety of strategies we can adopt:

- **Ignore these locations** — this means that we report only values for which every required image location exists. This has the advantage of probity, but the disadvantage that the output is smaller than the input — repeated convolutions can cause the image to shrink quite drastically.
- **Pad the image with constant values** — this means that, as we look at output values closer to the edge of the image, the extent to which the output of the convolution depends on the image goes down. This is a convenient trick, because we can ensure that the image doesn't shrink, but it has the disadvantage that it can create the appearance of substantial gradients near the boundary.
- **Pad the image in some other way** — for example, we might think of the image as a doubly periodic function, so that, if we have an $n \times m$ image, then column $m + 1$ — required for the purposes of convolution — would be the same as column $m - 1$. This can create the appearance of substantial second derivative values near the boundary.

7.3 Spatial Frequency and Fourier Transforms

We have used the trick of thinking of a signal $g(x, y)$ an weighted sum of a very large (or infinite) number of very small (or infinitely small) box functions. This model emphasizes that a signal is an element of a vector space — the box functions form a convenient basis, and the weights are coefficients on this basis. We need a new technique to deal with two related problems, so far left open:

- while it is clear that a discrete image version cannot represent the full information in a signal, we have not yet indicated what is lost;
- it is clear that we cannot shrink an image simply by taking every k'th pixel — this could turn a checkerboard image all white or all black — and we should like to know how to shrink an image safely.

All of these problems are related to the presence of fast changes in an image. For example, shrinking an image is most likely to miss fast effects, because they could slip between samples; similarly, the derivative is large at fast changes.

These effects can be studied by *a change of basis.* We will change the basis to be a set of sinusoids, and represent the signal as an infinite weighted sum of an infinite number of sinusoids. This means that fast changes in the signal will be obvious, because they will correspond to large amounts of high frequency sinusoids in the new basis.

7.3.1 Fourier Transforms

We will concentrate on Fourier transforms in the continuous domain, because we use Fourier transforms mainly as a conceptual device.

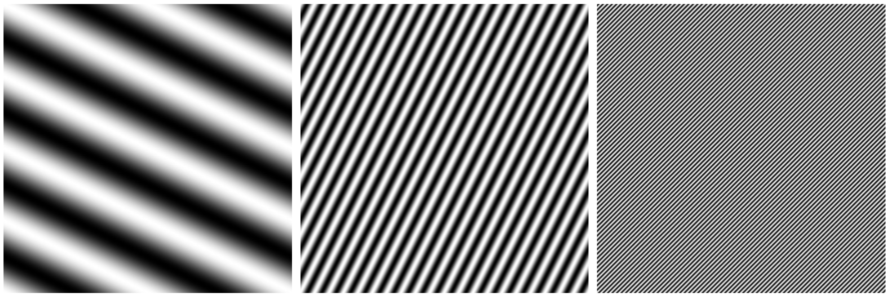


Figure 7.4. The real component of Fourier basis elements, shown as intensity images. The brightest point has value one, and the darkest point has value zero. The domain is $[-1,1]x[-1,1]$, with the origin at the center of the image. On the left, $(u,v) = (1,2)$; in the center, $(u,v) = (10,-5)$ and on the right $(u,v) = (32,-32)$. These are sinusoids of various frequencies and orientations, described in the text.

Fourier Transforms in the Continuous Domain

The change of basis is effected by a **Fourier Transform**. We define the Fourier transform of a signal $g(x,y)$ to be:

$$\mathcal{F}(g(x,y))(u,v) = \int\int_{-\infty}^{\infty} g(x,y)e^{-i2\pi(ux+vy)}dxdy$$

Assume that appropriate technical conditions are true to make this integral exist (it is sufficient for all moments of g to be finite; a variety of other possible conditions are available []). The process takes a complex valued function of x, y and returns a complex valued function of u, v (images are complex valued functions with zero imaginary component).

For the moment, fix u and v, and let us consider the meaning of the value of the transform at that point. The exponential can be rewritten

$$e^{-i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) + i\sin(2\pi(ux+vy))$$

These terms are sinusoids on the x, y plane, whose orientation and frequency are given by u, v. For example, consider the real term, which will be constant when $ux + vy$ is constant, i.e. along a straight line in the x, y plane whose orientation is given by $\tan\theta = v/u$. The gradient of this term is perpendicular to lines where $ux + vy$ is constant, and the frequency of the sinusoid is $\sqrt{u^2+v^2}$. These sinusoids are often referred to as **spatial frequency components**; a variety are illustrated in figure 7.4.

The integral should be seen as a dot product. If we fix u and v, the value of the integral is the dot product between a sinusoid in x and y and the original function. This is a useful analogy, because dot products measure the "amount" of one vector in the direction of another. In the same way, the value of the transform at a particular u and v can be seen as measuring the "amount" of the sinusoid with given frequency and orientation in the signal. Our transform is a record of this value for every u and v. The Fourier transform is linear:

$$\begin{aligned} \mathcal{F}(g(x,y)+h(x,y)) &= \mathcal{F}(g(x,y))+\mathcal{F}(h(x,y)) \\ &\text{and} \\ \mathcal{F}(kg(x,y)) &= k\mathcal{F}(g(x,y)) \end{aligned}$$

The Inverse Fourier Transform

It is useful to be able to recover a signal from its Fourier transform. This is another change of basis, with the form

$$g(x,y) = \iint\limits_{-\infty}^{\infty} \mathcal{F}(g(x,y))(u,v)e^{i2\pi(ux+vy)}dxdy$$

Fourier Transform Pairs

Fourier transforms are known in closed form for a variety of useful cases; a large set of examples appears in [**?**]. We list a few in table 7.1 for reference. The last line of table 7.1 contains the **convolution theorem**; convolution in the signal domain is the same as multiplication in the Fourier domain. We will use this important fact several times in what follows.

Phase and Magnitude

The Fourier transform consists of a real and a complex component.

$$\begin{aligned} \mathcal{F}(g(x,y))(u,v) &= \int\int_{-\infty}^{\infty} g(x,y)\cos(2\pi(ux+vy))dxdy + \\ &\quad i\int\int_{-\infty}^{\infty} g(x,y)\sin(2\pi(ux+vy))dxdy \\ &= \Re(\mathcal{F}(g)) + i * \Im(\mathcal{F}(g)) \\ &= \mathcal{F}_R(g) + i * \mathcal{F}_I(g) \end{aligned}$$

It is usually inconvenient to draw complex functions of the plane. One solution is to plot $\mathcal{F}_R(g)$ and $\mathcal{F}_I(g)$ separately; another is to consider the *magnitude* and *phase* of the complex functions, and to plot these instead. These are then called the **magnitude spectrum** and **phase spectrum** respectively.

The value of the Fourier transform of a function at a particular u, v point depends on the whole function. This is obvious from the definition, because the domain of the integral is the whole domain of the function. It leads to some subtle

Function	Fourier transform
$g(x,y)$	$\iint\limits_{-\infty}^{\infty} g(x,y)e^{-i2\pi(ux+vy)}dxdy$
$\delta(x,y)$	1
$\frac{\partial f}{\partial x}(x,y)$	$u\mathcal{F}(f)(u,v)$
$0.5\delta(x+a,y)+0.5\delta(x-a,y)$	$\cos 2\pi a u$
$e^{-\pi(x^2+y^2)}$	$e^{-\pi(u^2+v^2)}$
$bar_1(x,y)$	$\frac{\sin u}{u}\frac{\sin v}{v}$
$f(ax,by)$	$\frac{\mathcal{F}(f)(u/a,v/b)}{ab}$
$\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i,y-j)$	$\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(u-i,v-j)$
$(f**g)(x,y)$	$\mathcal{F}(f)\mathcal{F}(g)(u,v)$
$f(x-a,y-b)$	$e^{-i2\pi(au+bv)}\mathcal{F}(f)$
$f(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$	$\mathcal{F}(f)(u\cos\theta - v\sin\theta, u\sin\theta + v\cos\theta)$

Table 7.1. A variety of functions of two dimensions, and their Fourier transforms. This table can be used in two directions (with appropriate substitutions for u, v and x, y), because the Fourier transform of the Fourier transform of a function is the function. Observant readers may suspect that the results on infinite sums of δ functions contradict the linearity of Fourier transforms; by careful inspection of limits, it is possible to show that they do not (see, for example []).

properties, however. Firstly, a local change in the function — for example, zeroing out a block of points — is going to lead to a change *at every point* in the Fourier transform. This means that the Fourier transform is quite difficult to use as a representation — for example, it might be very difficult to tell if a pattern was present in an image just by looking at the Fourier transform. Secondly, the magnitude spectra of images tends to be similar — this appears to be a fact of nature, rather than something that can be proven axiomatically — and the magnitude spectrum of an image is surprisingly uninformative (see figure 7.5 for an example).

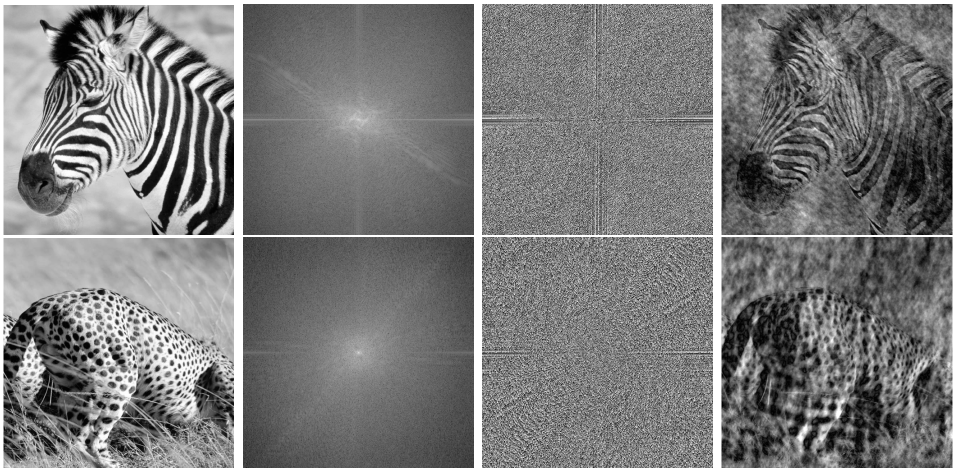


Figure 7.5. The second image in each row shows the log of the magnitude spectrum for the first image in the row; the third image shows the phase spectrum, scaled so that $-\pi$ is dark and π is light. The final images are obtained by swapping the magnitude spectra. While this swap leads to substantial image noise, it doesn't substantially affect the interpretation of the image, suggesting that the phase spectrum is more important for perception than the magnitude spectrum.

7.4 Sampling and Aliasing

The crucial reason to discuss Fourier transforms is to get some insight into the difference between discrete and continuous images; in particular, it is clear that some information has been lost when we work on a discrete pixel grid — but what? A good, simple example to think about is an image of a checkerboard. Assume that the checks are two units on edge. If we sample this image on a unit grid, we can ensure that there are four white pixels to each white check and four black pixels to each black one (assuming that we set up the grid in the right place and make sensible decisions about the colour value at the boundary of each check). Again, if we place samples two units apart, we can ensure that each check receives one pixel. Now consider what happens if the samples are *three* units apart — we can actually end up with four white pixels to each white check and four black pixels to each black one again, which is a significant misrepresentation. Things get worse if we place the samples four units apart — we could end up with an entirely white or an entirely black grid. The problem appears to have to do with the number of samples relative to the function; we can formalize this rather precisely, given a sufficiently powerful model.

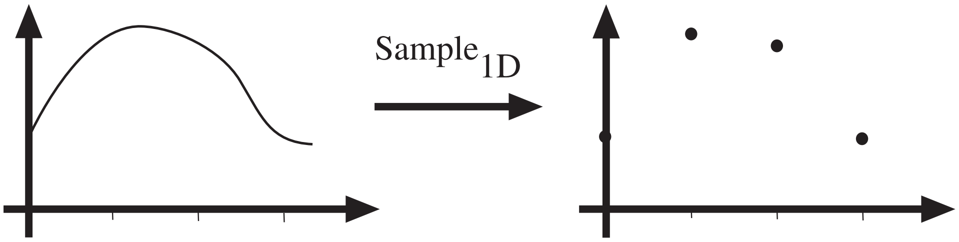


Figure 7.6. Sampling in 1D takes a function, and returns a vector whose elements are values of that function at the sample points, as the top figures show. For our purposes, it is enough that the sample points be integer values of the argument. We allow the vector to be infinite dimensional, and have negative as well as positive indices.

7.4.1 Sampling

Passing from a continuous function — like the irradiance at the back of a camera system — to a collection of values on a discrete grid — like the pixel values reported by a camera — is referred to as **sampling**. We will construct a model that allows us to obtain a precise notion of what is lost in sampling.

Sampling in One Dimension

Sampling in one dimension takes a function, and returns a discrete set of values. The most important case involves sampling on a uniform discrete grid, and we shall assume that the samples are defined at integer points. This means we have a process that takes some function and returns a vector of values:

$$\texttt{sample}_{1D}(f(x)) = \boldsymbol{f}$$

We will model this sampling process by assuming that the elements of this vector are the values of the function $f(x)$ at the sample points, and allowing negative indices to the vector. This means that the i'th component of $\boldsymbol{f}$ is $f(x_i)$.

Sampling in Two Dimensions

Sampling in 2D is very similar to sampling in 1D. Although sampling can occur on non-regular grids (the best example being the human retina), we will proceed on the assumption that samples are drawn at points with integer coordinates. This yields a uniform rectangular grid, which is a good model of most cameras. Our sampled images are then rectangular arrays of finite size (all values outside the grid being zero).

In the formal model, we sample a function of two dimensions, instead of one, yielding an array. This array we allow to have negative indices in both dimensions, and can then write

$$\texttt{sample}_{2D}(F(x, y)) = \mathcal{F}$$

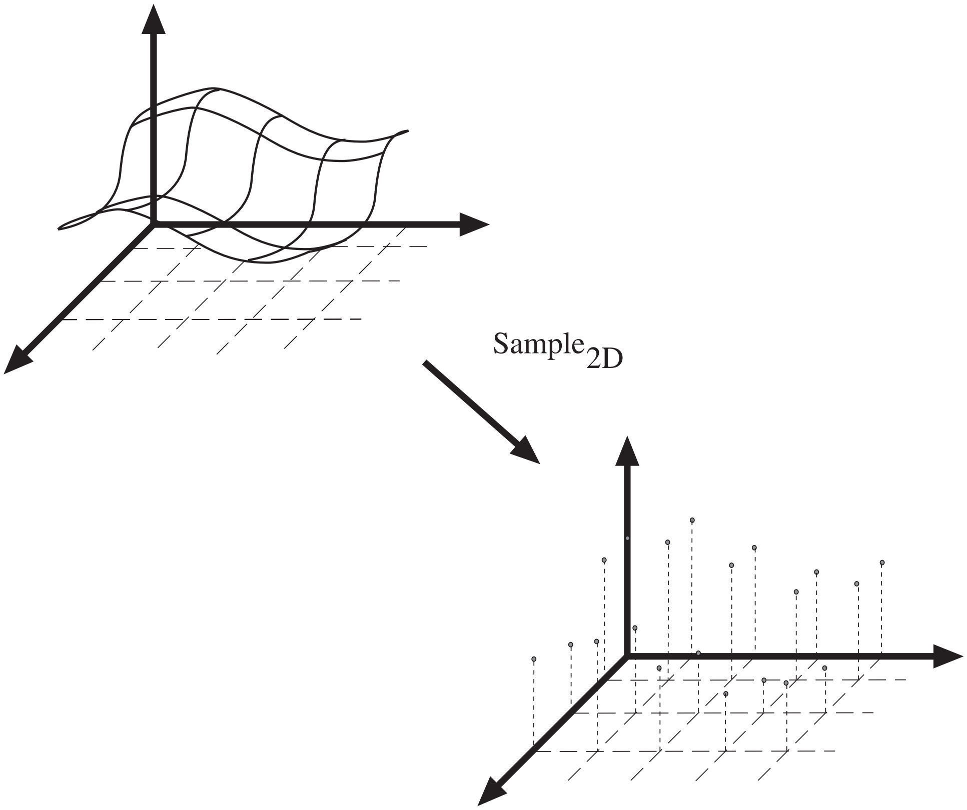


Figure 7.7. Sampling in 2D takes a function and returns an array; again, we allow the array to be infinite dimensional and to have negative as well as positive indices.

where the i, j'th element of the array $\mathcal{F}$ is $F(x_i, y_j) = F(i, j)$.

Samples are not always evenly spaced in practical systems. This is quite often due to the pervasive effect of television; television screens have an aspect ratio of 4:3 (width:height). Cameras quite often accomodate this effect by spacing sample points slightly further apart horizontally than vertically (in jargon, they have **non-square pixels**).

A Continuous Model of a Sampled Signal

We need a continuous model of a sampled signal. Generally, this model will be used to evaluate integrals — in particular, taking a Fourier transform involves integrating the product of our model with a complex exponential. It is clear how this integral should behave — the value of the integral should be obtained by adding up values at each integer point. This means we cannot model a sampled signal as a function that is zero everywhere except at integer points (where it takes the value of the

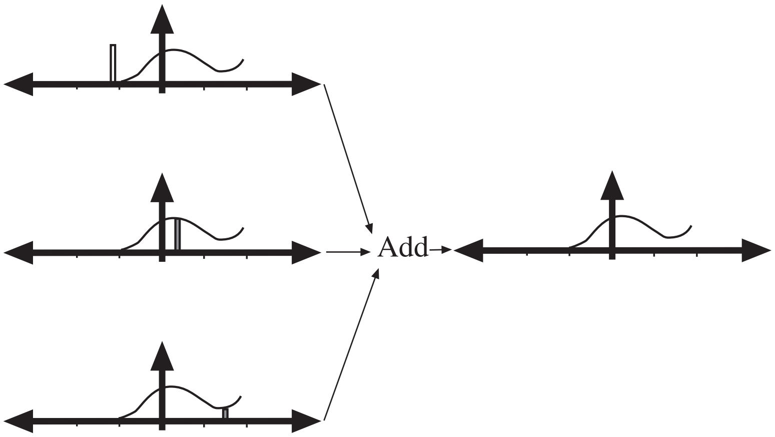


Figure 7.8. Convolving a δ-function with an arbitrary function can be thought of in terms of convolving a d_ϵ function, and taking the limit. The shaded region contributes to the integral, and as ϵ gets smaller — and so the region averaged gets narrower — the result limits to the original function.

signal), because this model has a zero integral.

An appropriate continuous model of a sampled signal relies on an important property of the δ function:

$$\begin{aligned}
\int_{-\infty}^{\infty} a\delta(x)f(x)dx &= a \lim_{\epsilon\to 0} \int_{-\infty}^{\infty} d(x;\epsilon)f(x)dx \\
&= a \lim_{\epsilon\to 0} \int_{-\infty}^{\infty} \frac{bar(x;\epsilon)}{\epsilon}(f(x))dx \\
&= a \lim_{\epsilon\to 0} \sum_{i=-\infty}^{\infty} \frac{bar(x;\epsilon)}{\epsilon}(f(i\epsilon)bar(x-i\epsilon;\epsilon))\epsilon \\
&= af(0)
\end{aligned}$$

(where we have used the idea of an integral as the limit of a sum of small strips).

An appropriate continuous model of a sampled signal consists of a δ-function at each sample point weighted by the value of the sample at that point. We can obtain this model by multiplying the sampled signal by a set of δ-functions, one at each sample point. In one dimension, a function of this form is called a **comb function** (because that's what the graph looks like). In two dimensions, a function of this form is called a **bed-of-nails function** (for the same reason).

Working in 2D and assuming that the samples are at integer points, this procedure gets

$$\begin{aligned}\texttt{sample}_{2D}(f) &= \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f(i,j)\delta(x-i,y-j)\\ &= f(x,y)\left\{\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i,y-j)\right\}\end{aligned}$$

This function is zero except at integer points (because the δ-function is zero except at integer points) and its integral is the sum of the function values at the integer points.

7.4.2 Aliasing

Sampling involves a loss of information. As this section will show, a signal that is sampled too slowly will be misrepresented by the samples; high spatial frequency components of the original signal will appear as low spatial frequency components in the sampled signal, an effect known as **aliasing**.

The Fourier Transform of a Sampled Signal

A sampled signal is given by a product of the original signal with a bed-of-nails function. By the convolution theorem, the Fourier transform of this product is the convolution of the Fourier transforms of the two functions. This means that the Fourier transform of a sampled signal is obtained by convolving the Fourier transform of the signal with another bed-of-nails function.

Now convolving a function with a shifted δ-function merely shifts the function (figure 7.8) (exercises). This means that the Fourier transform of the sampled signal is the sum of a collection of shifted versions of the Fourier transforms of the signal.

$$\begin{aligned}\mathcal{F}(\texttt{sample}_{2D}(f(x,y))) &= \mathcal{F}\left(f(x,y)\left\{\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i,y-j)\right\}\right) &(7.4.1)\\ &= \mathcal{F}(f(x,y)) **\mathcal{F}\left(\left\{\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\delta(x-i,y-j)\right\}\right) &(7.4.2)\\ &= \sum_{i=-\infty}^{\infty} F(u-i,v-j) &(7.4.3)\end{aligned}$$

where we have written the Fourier transform of $f(x,y)$ as $F(u,v)$.

If the support of these shifted versions of the Fourier transform of the signal does not intersect, we can easily reconstruct the signal from the sampled version. We take the sampled signal, Fourier transform it, and cut out one copy of the Fourier transform of the signal, and Fourier transform this back (figure 7.9).

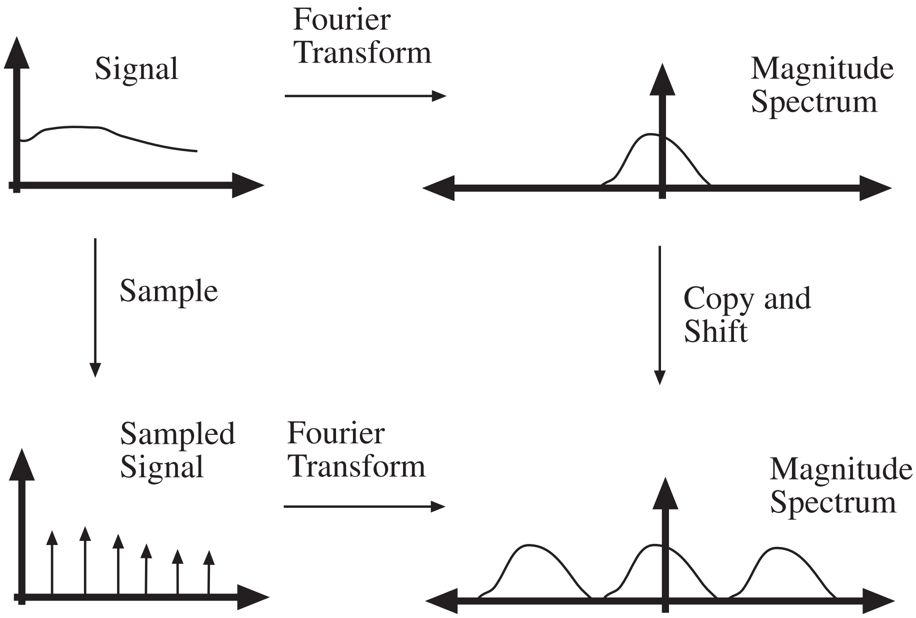


Figure 7.9. The Fourier transform of the sampled signal consists of a sum of copies of the Fourier transform of the original signal, shifted with respect to each other by the sampling frequency. Two possibilities occur. If the shifted copies do not intersect with each other (as in this case), the original signal can be reconstructed from the sampled signal (we just cut out one copy of the Fourier transform, and inverse transform it). If they do intersect (as figure 7.10) the intersection region is added, and so we cannot obtain a separate copy of the Fourier transform, and the signal has aliased.

However, if the support regions *do* overlap, we will not be able to reconstruct the signal because we will not be able to determine the Fourier transform of the signal in the regions of overlap, where different copies of the Fourier transform will add. This results in a characteristic effect, usually called **aliasing**, where high spatial frequencies appear to be low spatial frequencies (see figure 7.11 and exercises). Our argument also yields **Nyquist's theorem** — the sampling frequency must be at least twice the highest frequency present for a signal to be reconstructed from a sampled version.

7.4.3 Smoothing and Resampling

Nyquist's theorem means it is dangerous to shrink an image by simply taking every k'th pixel (as figure 7.11 confirms). Instead, we need to filter the image so that spatial frequencies above the new sampling frequency are removed. We could do this exactly by multiplying the image Fourier transform by a scaled 2D bar function, which would act as a low pass filter. Equivalently, we would convolve the image

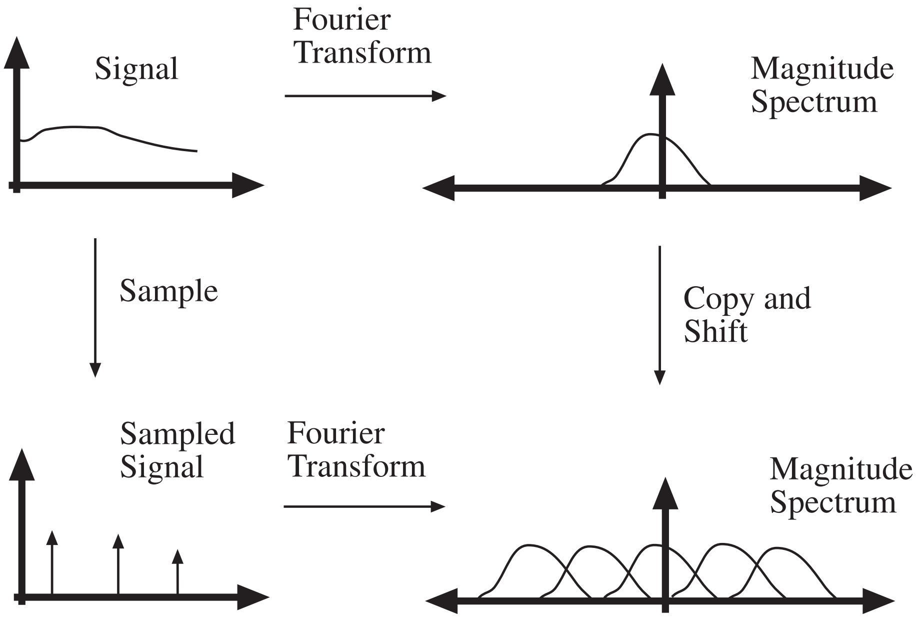


Figure 7.10. The Fourier transform of the sampled signal consists of a sum of copies of the Fourier transform of the original signal, shifted with respect to each other by the sampling frequency. Two possibilities occur. If the shifted copies do not intersect with each other (as in figure 7.9), the original signal can be reconstructed from the sampled signal (we just cut out one copy of the Fourier transform, and inverse transform it). If they do intersect (as in this figure) the intersection region is added, and so we cannot obtain a separate copy of the Fourier transform, and the signal has aliased. This also explains the tendency of high spatial frequencies to alias to lower spatial frequencies.

with a kernel of the form $(\sin x \sin y)/(xy)$. This is a difficult and expensive (a polite way of saying "impossible") convolution, because this function has infinite support.

The most interesting case occurs when we want to halve the width and height of the image. We assume that the sampled image has no aliasing (because if it did, there would be nothing we could do about it, anyway — once an image has been sampled, any aliasing that is going to occur has happened, and there's not much we can do about it without an image model). This means that the Fourier transform of the sampled image is going to consist of a set of copies of some Fourier transform, with centers shifted to integer points in u, v space.

If we resample this signal, the copies will now have centers on the half-integer points in u, v space. This means that, to avoid aliasing, we need to apply a low pass filter that strongly reduces the content of the original Fourier transform outside the range $|u| < 1/2$, $|v| < 1/2$. Of course, if we reduce the content of the signal *inside* this range, we may lose information, too.

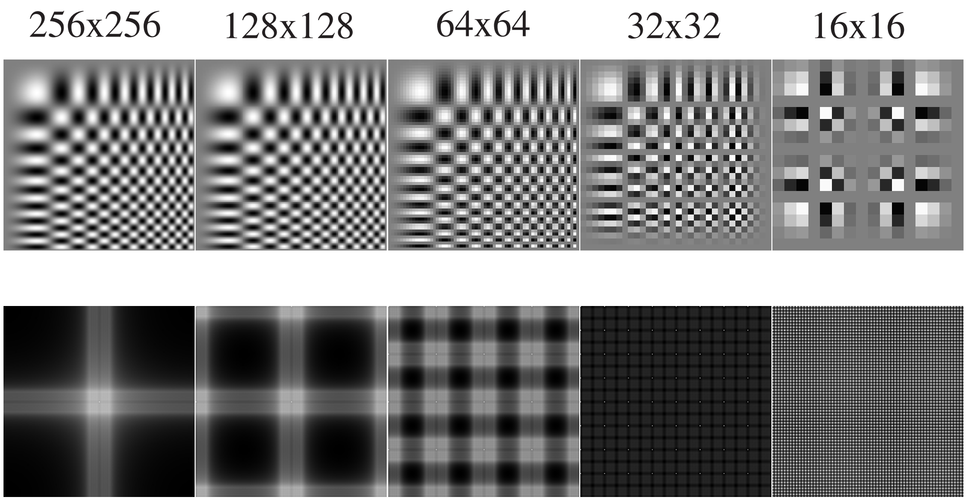


Figure 7.11. **Left:** At the top is a 256x256 pixel image showing a grid obtained by multiplying two sinusoids with linearly increasing frequency — one in x and one in y. The other images in the series are obtained by resampling by factors of two, without smoothing (i.e. the next is a 128x128, then a 64x64, etc., all scaled to the same size). Note the substantial aliasing; high spatial frequencies alias down to low spatial frequencies, and the smallest image is an extremely poor representation of the large image. **Right:** The magnitude of the Fourier transform of each image — displayed as a log, to compress the intensity scale. The constant component is at the center. Notice that the Fourier transform of a resampled image is obtained by scaling the Fourier transform of the original image and then tiling the plane. Interference between copies of the original Fourier transform means that we cannot recover its value at some points — this is the mechanism of aliasing.

Gaussians die away fairly quickly and so can be used as low pass filters. The choice of Gaussian depends on the application; if σ is large, the kernel must be large and, while there is less aliasing (because the value of the kernel outside our range is very small), information is lost because the kernel is not flat within our range; similarly, if σ is small, less information is lost within the range, but aliasing can be more substantial. Figures **??** and **??** illustrate the effects of different choices of σ.

7.5 Technique: Scale and Image Pyramids

Images look quite different at different scales. For example, the zebra's nose in figure **??** can be described in terms of individual hairs — which might be coded in terms of the response of oriented filters that operate at a scale of a small number of pixels — or in terms of the stripes on the zebra.

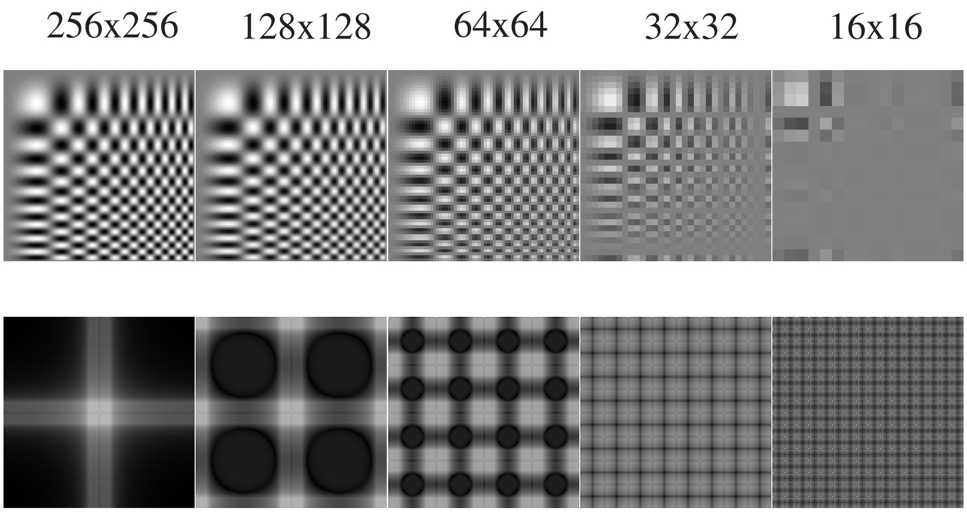


Figure 7.12. Left: Resampled versions of the image of figure 7.11, again by factors of two, but this time each image is smoothed with a Gaussian of σ one pixel before resampling. This filter is a low-pass filter, and so suppresses high spatial frequency components, reducing aliasing. **Right:** The effect of the low-pass filter is easily seen in these log-magnitude images; the low pass filter suppresses the high spatial frequency components so that components interfere less, to reduce aliasing.

7.5.1 The Gaussian Pyramid

A **pyramid** is a collection of representations of an image. The name pyramid comes from a visual analogy. Typically, each layer of the pyramid is half the width and half the height of the previous layer, and if we were to stack the layers on top of each other a pyramid would result. In a **Gaussian pyramid**, each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer (figure 7.14). These pyramids are most convenient if the image dimensions are a power of two, or a multiple of a power of two. The smallest image is the most heavily smoothed; the layers are often referred to as **coarse scale** versions of the image.

There are a variety of other kinds of pyramid, best understood with the aid of a little notation. To simplify things, assume that the original image is square, with image dimensions 2^k. The operator $S^{\downarrow}$ downsamples an image; in particular, the j, k'th element of $S^{\downarrow}(\mathcal{I})$ is the $2j$, $2k$'th element of $\mathcal{I}$. The n'th level of a pyramid $P(\mathcal{I})$ is denoted $P(\mathcal{I})_n$.

We can now write simple expressions for the layers of a Gaussian pyramid:

$$
\begin{aligned}
P_{\text{Gaussian}}(\mathcal{I})_{n+1} &= S^{\downarrow}(G_{\sigma} ** P_{\text{Gaussian}}(\mathcal{I})_n) && (7.5.1)\\
&= (S^{\downarrow}G_{\sigma})P_{\text{Gaussian}}(\mathcal{I})_n) && (7.5.2)
\end{aligned}
$$

(where we have written G_{σ} for the linear operator that takes an image to the

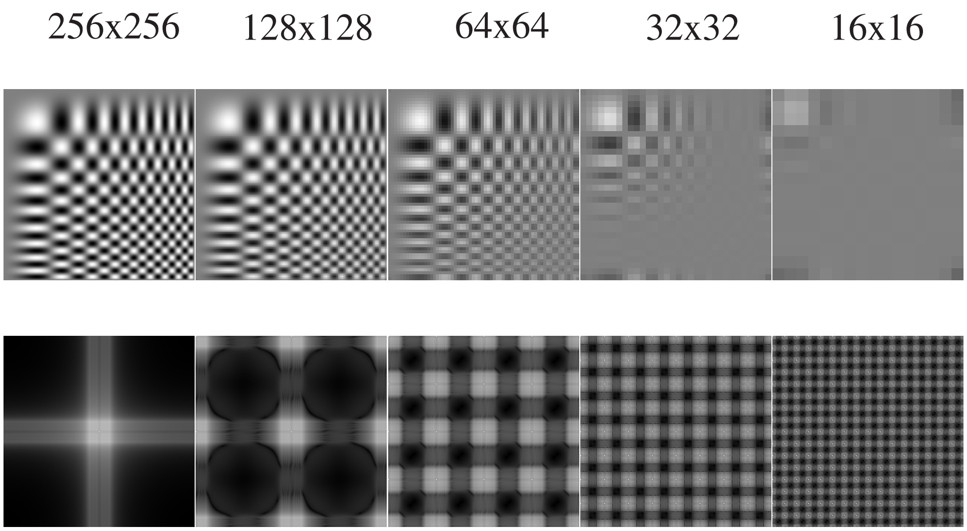


Figure 7.13. **Left:** Resampled versions of the image of figure 7.11, again by factors of two, but this time each image is smoothed with a Gaussian of σ two pixels before resampling. This filter suppresses high spatial frequency components more aggressively than that of figure 7.12. **Right:** The effect of the low-pass filter is easily seen in these log-magnitude images; the low pass filter suppresses the high spatial frequency components so that components interfere less, to reduce aliasing.

convolution of that image with a Gaussian). The finest scale layer is the original image

$$P_{\text{Gaussian}}(\mathcal{I})_1 = \mathcal{I}$$

A simple, immediate use for a Gaussian pyramid is to obtain zero-crossings of a Laplacian of Gaussian (or a DOG) at various levels of smoothing.

7.5.2 Applications of Scaled Representations

Gaussian pyramids are useful, because they make it possible to extract representations of different types of structure in an image. For example, in the case of the zebra, we would not want to apply very large filters to find the stripes. This is because these filters are inclined to spurious precision — we don't wish to have to represent the disposition of each hair on the stripe — inconvenient to build, and slow to apply.

A more practical approach than applying very large filters is to apply smaller filters to a less detailed version of the image. We expect effects to appear at a variety of scales, and so represent the image in terms of several smoothed and sampled versions. Continuing with the zebra analogy, this means we can represent the hair on the animal's nose, the stripes of its body, dark legs, and entire zebras,

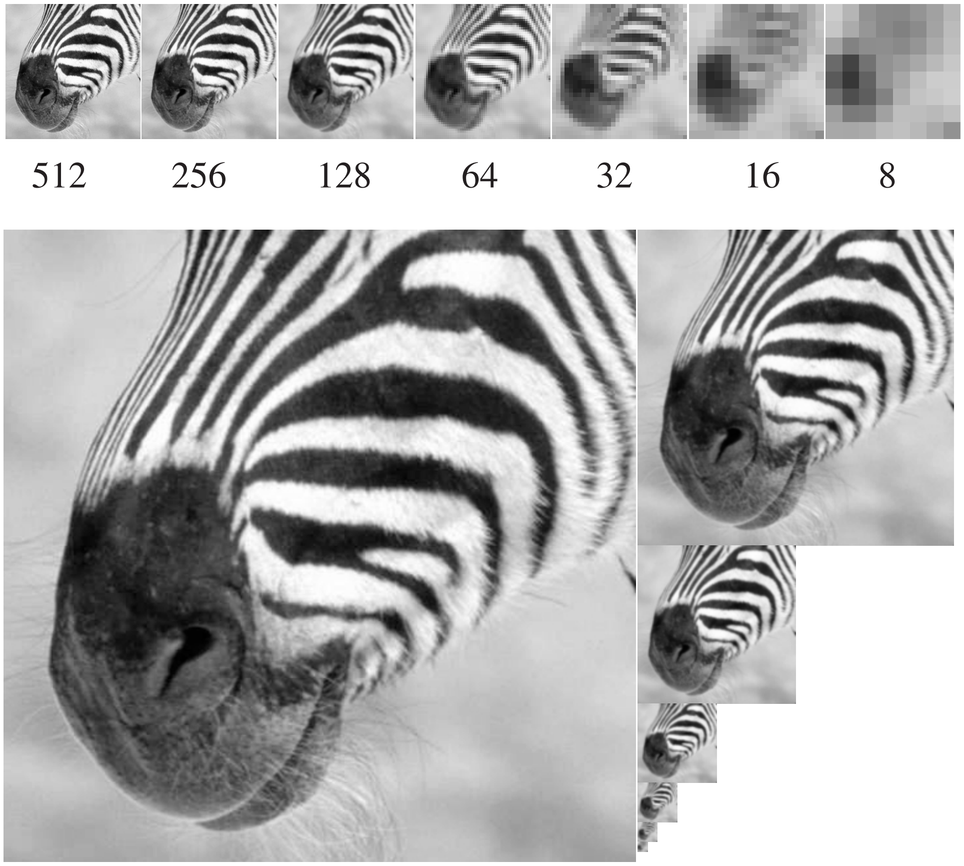


Figure 7.14. A Gaussian pyramid of images, running from 512x512 to 8x8. On the top row, we have shown each image at the same size (so that some have bigger pixels than others), and the lower part of the figure shows the images to scale.

each as bars of different sizes.

Scale and Spatial Search

Another application is spatial search, a common theme in computer vision. Typically, we have a point in one image and are trying to find a point in a second image that corresponds to it. This problem occurs in stereopsis — where the point has moved because the two images are obtained from different viewing positions — and in motion analysis — where the image point has moved either because the camera moved, or because it is on a moving object.

Searching for a match in the original pairs of images is inefficient, because we may have to wade through a great deal of detail. A better approach, which is now

```
Set the finest scale layer to the image

For each layer, going from next to finest to coarsest

  Obtain this layer by smoothing the next finest
  layer with a Gaussian, and then subsampling it

end
```

Algorithm 7.1: *Forming a Gaussian pyramid*

pretty much universal, is to look for a match in a heavily smoothed and resampled image, and then refine that match by looking at increasingly detailed versions of the image. For example, we might reduce 1024x1024 images down to 4x4 versions, match those and then look at 8x8 versions (because we know a rough match, it is easy to refine it); we then look at 16x16 versions, etc. all the way up to 1024x1024. This gives an extremely efficient search, because a step of a single pixel in the 4x4 version is equivalent to a step of 256 pixels in the 1024x1024 version. We will explore this strategy of **coarse-to-fine matching** in greater detail in chapters **??**.

Feature Tracking

Most features found at coarse levels of smoothing tend are associated with large, high contrast image events, because for an feature to be marked at a coarse scale a large pool of pixels need to agree that it is there. Typically, these phenomena misestimate the size of a feature — the contrast might decay along an edge, for example — and their accuracy can be quite poor — a single pixel error in a coarse-scale image represents a multiple pixel error in a fine-scale image.

At fine scales, there are many features, some of which are associated with smaller, low contrast events. One strategy for improving a set of features obtained at a fine scale is to track features across scales to a coarser scale, and accept only the fine scale features that have identifiable parents at a coarser scale. This strategy in principle can suppress features resulting from textured regions (often referred to as "noise") and features resulting from real noise.

7.5.3 Scale Space

Coarse scale components give the overall structure of a signal, and fine scale components give detailed information, as figure **??** suggests. This approach allows us to think about representing such objects as trees, which appear to exist at several distinct scales; we would want to be able to represent a tree both as a puff of foliage

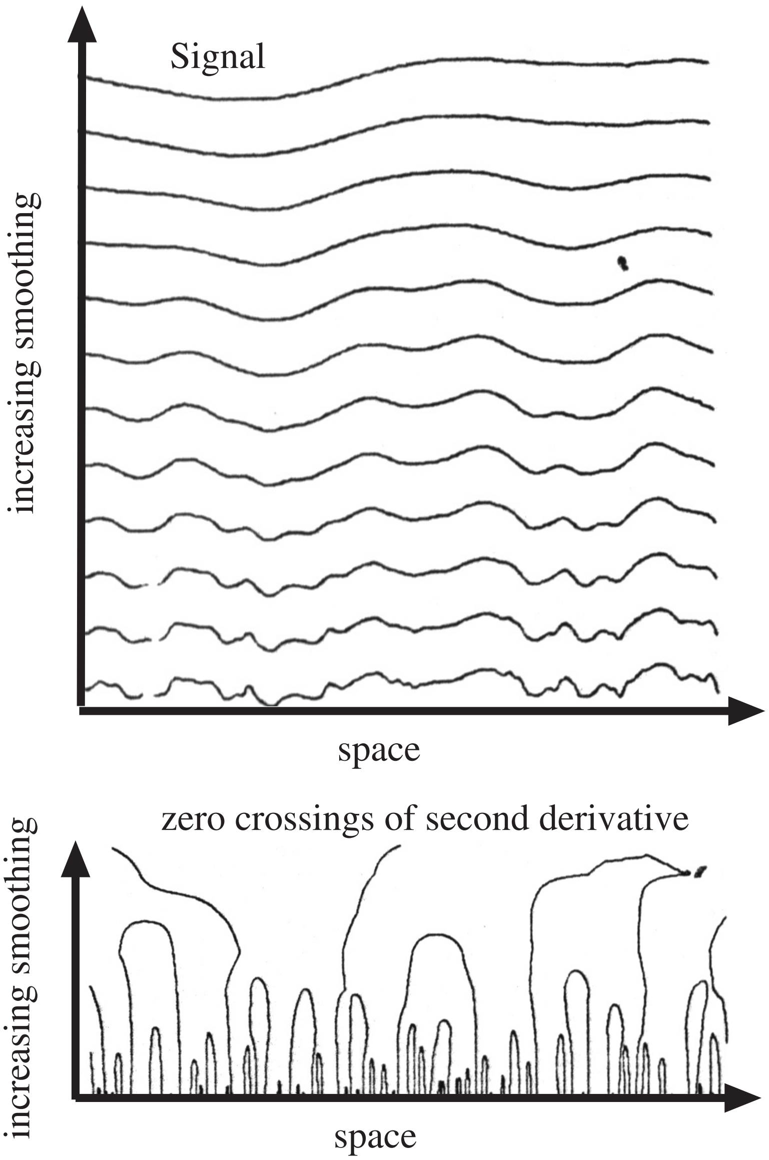


Figure 7.15. On the left, a 1D signal smoothed using Gaussian filters of increasing σ. As the signal is smoothed, extrema merge and vanish. The smoothest versions of the signal can be seen as an indication of the "overall trend" of the signal, and the finer versions have increasing amounts of detail imposed. The representation below marks the position of zero crossings of the second derivative of the smoothed signal, as the smoothing increases (again, scale increases vertically). Notice that zero crossings can meet and obliterate one another as the signal is smoothed but no new zero-crossing is created. This means that the figure shows the characteristic structure of either vertical curves or inverted "u" curves. An inverted pitchfork shape is also possible — where three extrema meet and become one — but this requires special properties of the signal and usually becomes an inverted u next to a vertical curve. Notice also that the position of zero crossings tends to shift as the signal is smoothed. *Figure from "Scale-space filtering," A.P. Witkin, Proc. 8'th IJCAI, 1983, page 1020, in the fervent hope, etc.*

on top of a stalk (coarse scale) and as a collection of individual leaves and twigs (fine scale).

Gaussian pyramids are not an ideal tool for this representation, because they sample the range of smoothed images quite coarsely. Instead of a discrete pyramid of images, we might consider a one parameter family of images (or, equivalently, a function of three dimensions)

$$\Phi(x, y, u) = G_{\sigma(u)} * \mathcal{I}(x, y)$$

where the extent of the smoothing is now a continuous parameter. For a 1D signal, we can draw the behaviour of features as the scaling parameter changes, and this drawing gives us a simple and sometimes quite informative description of the signal (figure **??**). If we define a "feature" to be a zero-crossing of the Laplacian, then it is possible to show that in this family of images, features are not created by smoothing. This means that all coarse-scale zero-crossings will have finer-scale events corresponding to them, so that there are quite simple rules for what these drawings look like (figure **??**). These drawings and the underlying representations are often referred to as **scale space**.

If we think of the signal as being composed of a set of parametric patches joined at these feature points, we obtain a different decomposition of the signal at each scale. Instead of representing the signal with a continuous family of decompositions where the feature points move, we could fix the location of these feature points and then use the simple rules for constructing scale-space drawings to obtain a strip-like decomposition (as in figure **??**). This is not a decomposition with a canonical form — we get to choose what the features are and the nature of the parametric patches — but it is often rather useful in practice, mainly because the decomposition of the signal appears to reflect the structure apparent to humans.

2D scale space

It is possible to extend these decompositions from 1D to 2D. Again, the choice of features is somewhat open, but a reasonable choice is the points of maximum or minimum brightness. Smoothing an image with a symmetric Gaussian cannot create local maxima or minima in brightness, but it can (and does) destroy them. Assume we have a scale value σ_{die} where a maximum (or minimum — we will just talk about the one case, for simplicity) is destroyed. If we now *reduce* the scale, a standard pattern will appear — there will be a corresponding maximum surrounded by a curve of equal brightness that has a self intersection. Thus, corresponding to each maximum (or minimum) at any scale, we have a **blob**, which is the region of the image marked out by this curve of equal brightness. Typically, maxima are represented by light blobs and minima by dark blobs. All of this gives us a representation of an image in terms of blobs growing (or dying) as the scale is decreased (or increased).

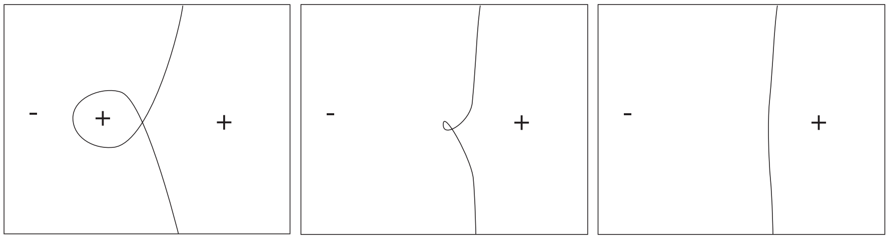


Figure 7.16. Smoothing an image with a symmetric Gaussian cannot create local maxima or local minima of brightness. However, local extrema can be extinguished. What happens is that a local maximum shrinks down to the value of the surrounding pixels. This is most easily conveyed by drawing a contour plot — the figure shows a contour plot with a local maximum which dies as the image is smoothed (the curve is a contour of constant brightness; the regions marked + have brightness greater than that of the contour, so the blob must have a maximum in it; the regions marked - have brightness less than that of the contour). Recording the details of these disappearances — where the maximum that disappears is, the contour defining the blob around the maximum, and the scale at which it disappears — yields a scale-space representation of the image.

7.6 Discussion

The book (hah! mere mathematics has a book - vision has a library) in which the secrets of vision are recorded must contain many volumes on the subject of filters; we have had to omit topics very aggressively to produce a survey of reasonable length. The topics whose omission will most disturb the informed are probably (i) the design of filters and (ii) techniques to obtain filtered representations efficiently. The first three sections of this chapter are essential if you want to study these topics; it wouldn't be possible to read the literature on filters in vision without these ideas.

7.6.1 Real Imaging Systems vs Shift-Invariant Linear Systems

Imaging systems are only approximately linear. Film is not linear — it does not respond to very weak stimuli, and it saturates for very bright stimuli — but one can usually get away with a linear model within a reasonable range. CCD cameras are linear within a working range. They give a very small, but non-zero response to a zero input, as a result of thermal noise (which is why astronomers cool their cameras) and they saturate for very bright stimuli. CCD cameras often contain electronics that transforms their output to make them behave more like film, because consumers are used to film. Shift invariance is approximate as well, because lenses tend to distort responses near the image boundary. Some lenses — fish-eye lenses are a good example — are not shift-invariant.

7.6.2 Scale

There is a very large body of work on scale space and scaled representations. We have given only the briefest picture here, because the analysis tends to be quite tricky. The usefulness of the techniques is currently hotly debated, too. The general idea is as follows: a representation of an object like a tree should probably occur at multiple scales — at a fine scale, we consider each twig and leaf, and at a coarse scale there are a couple of puffs of leaves at the top of a trunk. One technique that might It is something of a stretch from a representation of the behaviour of brightness maxima/minima to (say) a description of a tree. Generally, high maxima and deep minima will give blobs that last over a large range of scales, so that dark leaves on a bright sky may lead to a very large collection of small blobs which slowly merge over scales to end up with a single dark blob (the puff at the top of the tree). Much of the detailed gymnastics of the blobs as they merge is irrelevant — we really care only about the statistics of the blobs at the finest scale, and the size of the blob at the coarsest. There is little formalised knowledge about which bits of the representation carry cogent information and which do not.

Assignments

Exercises

1. Show that forming unweighted local averages — which yields an operation of the form

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}$$

   is a convolution. What is the kernel of this convolution?

2. Write $\mathcal{E}_0$ for an image that consists of all zeros, with a single one at the center. Show that convolving this image with the kernel

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{((i-k-1)^2 + (j-k-1)^2)}{2\sigma^2}\right)$$

   (which is a discretised Gaussian) yields a circularly symmetric fuzzy blob.

3. Show that convolving an image with a discrete, separable 2D filter kernel is equivalent to convolving with two 1D filter kernels. Estimate the number of operations saved for an NxN image and a $2k+1 \times 2k+1$ kernel.

4. Show that convolving a function with a shifted δ function shifts the function.

5. Aliasing takes high spatial frequencies to low spatial frequencies. Explain why following effects occur:

   - In old cowboy films that show wagons moving, the wheel often seems to be stationary or moving in the wrong direction (i.e. the wagon moves from left to right and the wheel seems to be turning anticlockwise).
   - White shirts with thin dark pinstripes often generate a shimmering array of colours on television.
   - In ray-traced pictures, soft shadows generated by area sources look blocky.

6. Each pixel value in 500 × 500 pixel image $\mathcal{I}$ is an independent normally distributed random variable with zero mean and standard deviation one. Estimate the number of pixels that where the absolute value of the x derivative, estimated by forward differences (i.e. $|I_{i+1,j} - I_{i,j}|$ is greater than 3.

7. Each pixel value in 500 × 500 pixel image $\mathcal{I}$ is an independent normally distributed random variable with zero mean and standard deviation one. $\mathcal{I}$ is convolved with the $2k + 1 \times 2k + 1$ kernel $\mathcal{G}$. What is the covariance of pixel values in the result? (There are two ways to do this; on a case-by-case basis — e.g. at points that are greater than $2k + 1$ apart in either the x or y direction, the values are clearly independent — or in one fell swoop. Don't worry about the pixel values at the boundary.)

Programming Assignments

- One way to obtain a Gaussian kernel is to convolve a constant kernel with itself, many times. Compare this strategy with evaluating a Gaussian kernel.
   1. How many repeated convolutions will you need to get a reasonable approximation? (you will need to establish what a reasonable approximation is; you might plot the quality of the approximation against the number of repeated convolutions).
   2. Are there any benefits that can be obtained like this? (hint: not every computer comes with an FPU)
- A sampled Gaussian kernel must alias, because the kernel contains components at arbitrarily high spatial frequencies. Assume that the kernel is sampled on an infinite grid. As the standard deviation gets smaller, the aliased energy must increase. Plot the energy that aliases against the standard deviation of the Gaussian kernel in pixels. Now assume that the Gaussian kernel is given on a 7x7 grid. If the aliased energy must be of the same order of magnitude as the error due to truncating the Gaussian, what is the smallest standard deviation that can be expressed on this grid?

Chapter 8

EDGE DETECTION

Sharp changes in image brightness are interesting for many reasons. Firstly, object boundaries often generate sharp changes in brightness — a light object may lie on a dark background, or a light object may lie on a dark background. Secondly, reflectance changes often generate sharp changes in brightness which can be quite distinctive — zebras have stripes and leopards have spots. Cast shadows can also generate sharp changes in brightness. Finally, sharp changes in surface orientation are often associated with sharp changes in image brightness.

Points in the image where brightness changes particularly sharply are often called **edges** or **edge points**. We should like edge points to be associated with the boundaries of objects and other kinds of meaningful changes. It is hard to define precisely the changes we would like to mark — is the region of a pastoral scene where the leaves give way to the sky the boundary of an object? Typically, it is hard to tell a semantically meaningful edge from a nuisance edge, and to do so requires a great deal of high-level information. Nonetheless, experience building vision systems suggests that very often, interesting things are happening in an image at an edge and it is worth knowing where the edges are.

We will proceed with a rather qualitative model of an edge as a point where the change of image brightness is distinctive and large. One sign of a sharp change in an image is a large gradient magnitude. We discuss estimating image derivatives in section 8.1 (which leads to a simple edge detector based on second derivative properties section 8.1.2), deal with the associated noise issues in section 8.6, and show how to to build an edge detector using gradient magnitude estimates in section 8.3.

8.1 Estimating Derivatives with Finite Differences

Estimates of derivatives are generally important in vision, because changes in images are interesting. A sharp change in an image could be associated with the boundary of an object, or with markings on the object; and such changes are associated with large gradients.

To estimate a derivative of an image represented by a discrete set of pixels, we need to resort to an approximation. Derivatives are rather naturally approximated

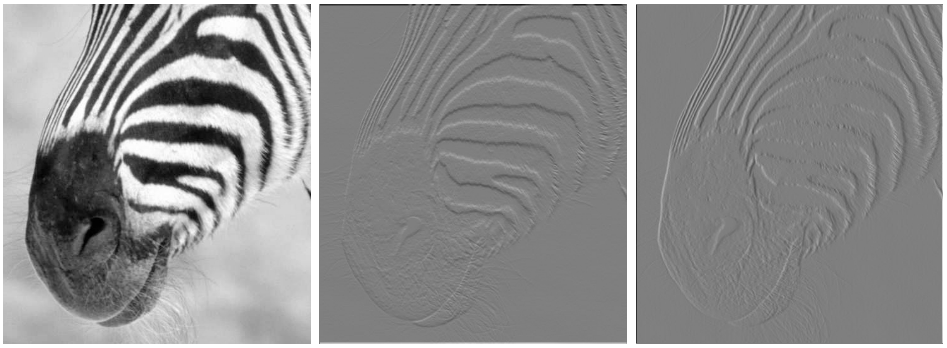


Figure 8.1. Finite differences are one way to obtain an estimate of a derivative. The image at the left shows a detail from a picture of a zebra. The center image shows the partial derivative in the y-direction — which responds strongly to horizontal stripes and weakly to vertical stripes — and the right image shows the partial derivative in the x-direction — which responds strongly to vertical stripes and weakly to horizontal stripes. In the derivative figures, a mid grey level is a zero value, a dark grey level is a negative value, and a light grey level is a positive value.

by **finite differences**. Because

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

we might estimate a partial derivative as a symmetric difference:

$$\frac{\partial h}{\partial x} \approx h_{i+1,j} - h_{i-1,j}$$

This is the same as a convolution, where the convolution kernel is

$$\mathcal{G} = \left\{ \begin{array}{ccr} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{array} \right\}$$

Notice that this kernel could be interpreted as a template — it will give a large positive response to an image configuration that is positive on one side and negative on the other, and a large negative response to the mirror image.

As figure 8.2 suggests, finite differences give a most unsatisfactory estimate of the derivative. This is because this filter has a strong response to fast changes due to noise, as well as those due to signal. For example, if we had bought a discount camera with some pixels that were stuck at either black or white, the filter would produce a strong response at those pixels because they will, in general, be substantially different from their neighbours. All this suggests that some form of smoothing is appropriate before differentiation; the details appear in section 8.3.1.

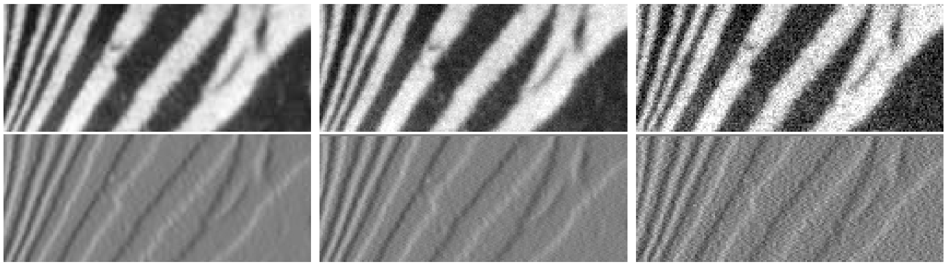


Figure 8.2. Finite differences respond strongly to noise. The image at top left shows a detail from a picture of a zebra; the next image in the row is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.03$) to each pixel; and the third image is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.09$) to each pixel. The second row shows the partial derivative in the x-direction of the image at the head of the row. Notice how strongly the differentiation process emphasizes image noise — the derivative figures look increasingly grainy. In the derivative figures, a mid grey level is a zero value, a dark grey level is a negative value, and a light grey level is a positive value.

8.1.1 Differentiation and Noise

From table 7.1, differentiating a function is the same as multiplying its Fourier transform by a frequency variable; this means that the high spatial frequency components are heavily emphasized at the expense of the low frequency components. This is intuitively plausible — differentiating a function must set the constant component to zero, and the amplitude of the derivative of a sinusoid goes up with its frequency. Furthermore, this property is the reason we are interested in derivatives; we are discussing the derivative precisely because fast changes (which generate high spatial frequencies) have large derivatives.

It is possible to show that stationary additive Gaussian noise has uniform energy at each frequency; but if we differentiate the noise, we will emphasize the high frequencies. If we do not attempt to ameliorate this situation, the gradient magnitude map is likely to have occasional large values due to noise. Filtering with a Gaussian filter suppresses these high spatial frequencies.

The discussion of aliasing gives us some insight into available smoothing parameters as well. Any Gaussian kernel that we use will be a sampled approximation to a Gaussian, sampled on a single pixel grid. This means that, for the original kernel to be reconstructed from the sampled approximation, it should contain no components of spatial frequency greater than 0.5pixel^{-1}. This isn't possible with a Gaussian, because its Fourier transform is also Gaussian, and hence isn't bandlimited. The best we can do is insist that the quantity of energy in the signal that is aliased is below some threshold — in turn, this implies a minimum value of σ that is available for a smoothing filter on a discrete grid (for values lower than this minimum, the smoothing filter itself is badly aliased — see exercise **??**).

8.1.2 Laplacians and edges

One diagnostic for a large gradient magnitude is a zero of a "second derivative" at a point where the gradient is large. A sensible 2D analogue to the 1D second derivative must be rotationally invariant; it is not hard to show that the **Laplacian** has this property. The Laplacian of a function in 2D is defined as:

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

It is natural to smooth the image It turns out that smoothing an image and then applying the Laplacian is the same as convolving the image with the Laplacian of a Gaussian (section 8.3.4 explains this fact for first derivatives, but it works for any linear differential operator).

This leads to a simple and historically important edge detection strategy, illustrated in figure 8.3. We convolve an image with a **Laplacian of Gaussian** at some scale, and mark the points where the result has value zero — the **zero crossings**. These points should be checked to ensure that the gradient magnitude is large.

The response of a Laplacian of Gaussian filter is positive on one side of an edge and negative on another. This means that adding some percentage of this response back to the original image yields a picture in which edges have been sharpened, and detail is more easy to see. This observation dates back to a photographic developing technique called *unsharp masking*, where a blurred positive is used to increase visibility of detail in bright areas by "subtracting" a local average of the brightness in that area. Unsharp masking essentially applies a difference of Gaussian kernel to an image; as exercise **??** indicates, the difference between two Gaussian kernels looks very similar to a Laplacian of Gaussian kernel. It is quite common to replace one with the other.

Laplacian of Gaussian edge detectors have fallen into some disfavour. Because the Laplacian of Gaussian filter is not oriented, its response is composed of an average across an edge and one along the edge. This means that their behaviour at corners — where the direction along the edge changes — is poor. They mark the boundaries of sharp corners quite inaccurately; furthermore, at trihedral or greater vertices, they have difficulty recording the topology of the corner correctly, as figure 8.4 illustrates. Secondly, the components along the edge tend to contribute to the response of the filter to noise but not necessarily to an edge; this means that zero-crossings may not lie exactly on an edge.

8.2 Noise

We have asserted that smoothing suppresses some kinds of noise. To be more precise we need a model of noise. Usually, by the term **noise**, we mean image measurements from which we do not know how to extract information, or from which we do not care to extract information; all the rest is **signal**. It is wrong to believe that noise does not contain information — for example, we should be able

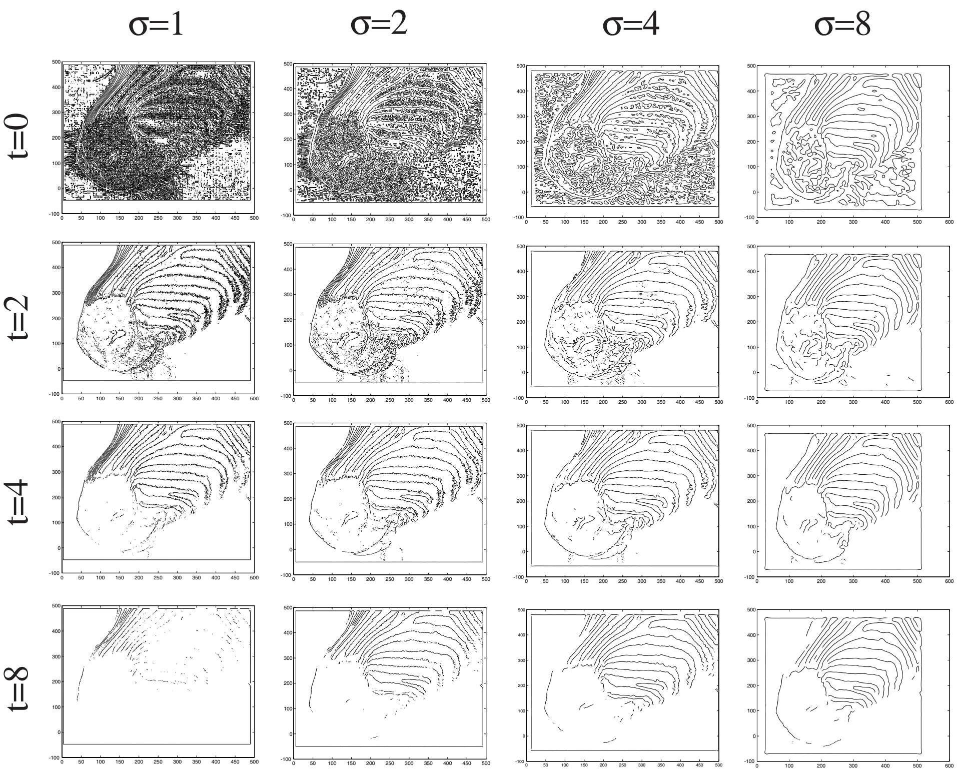


Figure 8.3. Zero crossings of the Laplacian of Gaussian for various scales and at various gradient magnitude thresholds. Each column shows a fixed scale, with t, the threshold on gradient magnitude increasing as one moves down the right (by a factor of two from image to image). Each row shows a fixed t, with scale increasing from σ one pixel to σ eight pixels, by factors of two. Notice that the fine scale, low threshold edges contain a quantity of detailed information that may or may not be useful (depending on one's interest in the hairs on the zebra's nose). As the scale increases, the detail is suppressed; as the threshold increases, small regions of edge drop out. No scale or threshold gives the outline of the zebra's head; all respond to its stripes, though as the scale increases, the narrow stripes on the top of the muzzle are no longer resolved.

to extract some estimate of the camera temperature by taking pictures in a dark room with the lens-cap on. Furthermore, since we cannot say anything meaningful about noise without a noise model, it is wrong to say that noise is not modelled. Noise is everything we don't wish to use, and that's all there is to it.

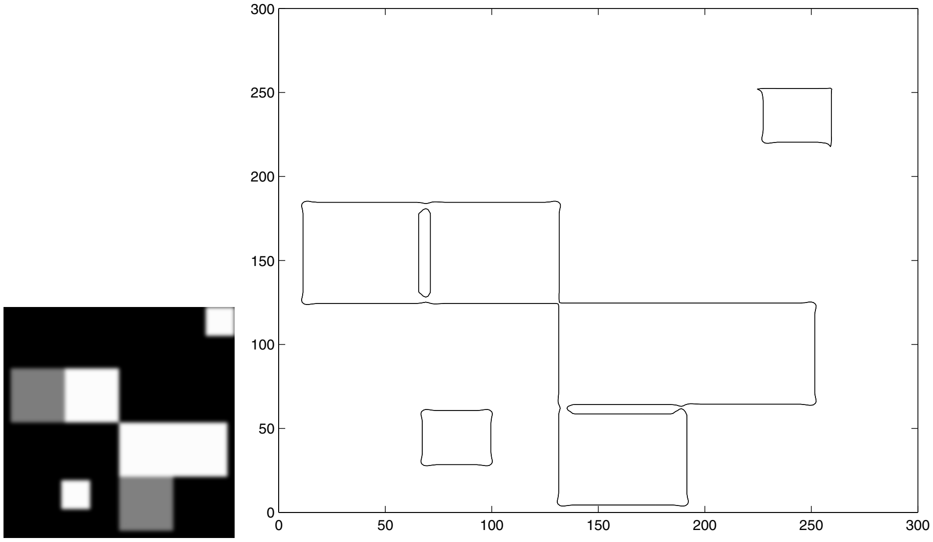


Figure 8.4. Zero crossings of Laplacian of Gaussian output can behave strangely at corners. Firstly, at a right angled corner, the zero crossing bulges out at the corner (but passes through the vertex). This effect is not due to digitisation or to quantization, but can be shown to occur in the continuous case as well. At corners where three or more edges meet, contours behave strangely, with the details depending on the structure of the contour marking algorithm — this algorithm (the one shipped with Matlab) produces curious loops. This effect can be mitigated with careful design of the contour marking process, which needs to incorporate a fairly detailed vertex model.

8.2.1 Additive Stationary Gaussian Noise

In the **additive stationary Gaussian noise** model, each pixel has added to it a value chosen independently from the same Gaussian probability distribution. Almost always, the mean of this distribution is zero. The standard deviation is a parameter of the model. The model is intended to describe thermal noise in cameras.

Linear Filter Response to Additive Gaussian Noise

Assume we have a discrete linear filter whose kernel is $\mathcal{G}$, and we apply it to a noise image $\mathcal{N}$ consisting of stationary additive Gaussian noise with mean μ and standard deviation σ. The response of the filter at some point i, j will be:

$$R(\mathcal{N})_{i,j} = \sum_{u,v} G_{i-u,j-v} N_{u,v}$$

Because the noise is stationary, the expectations that we compute will not depend on the point, and we assume that i and j are zero, and dispense with the

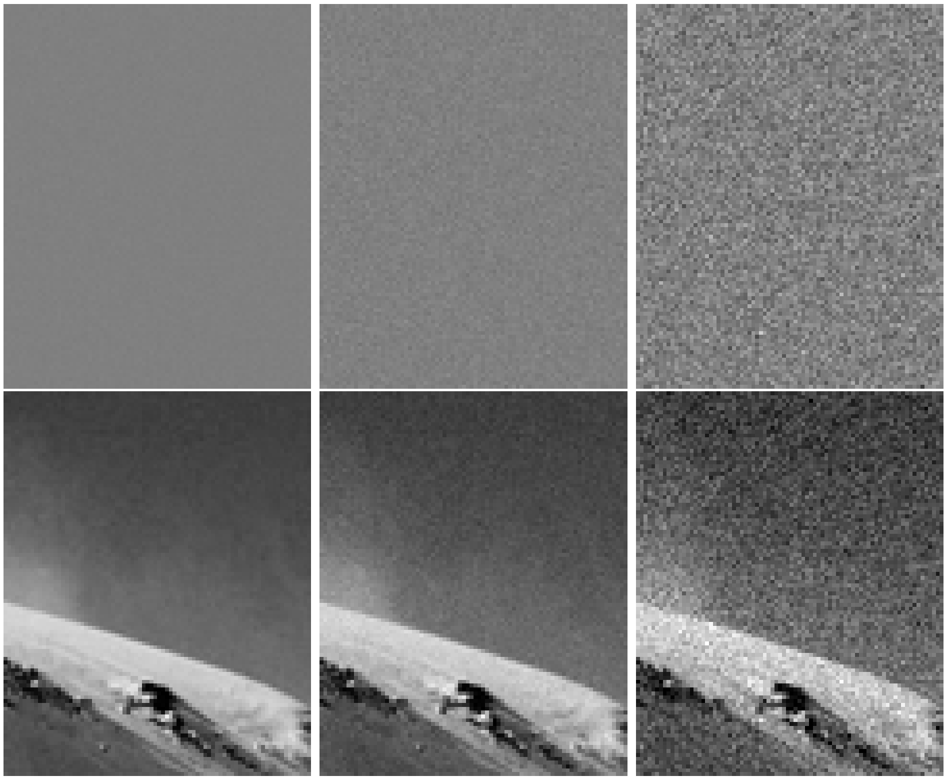


Figure 8.5. The top row shows three realisations of a stationary additive Gaussian noise process. We have added half the range of brightnesses to these images, so as to show both negative and positive values of noise. From left to right, the noise has standard deviation 1/256, 4/256 and 16/256 of the full range of brightness respectively. This corresponds roughly to bits zero, two and five of a camera that has an output range of eight bits per pixel. The lower row shows this noise added to an image. In each case, values below zero or above the full range have been adjusted to zero or the maximum value accordingly.

subscript. Assume the kernel has finite support, so that only some subset of the noise variables contributes to the expectation; write this subset as $n_{0,0}, \ldots, n_{r,s}$. The expected value of this response must be:

$$\begin{aligned}
\mathtt{E}[R(\mathcal{N})] &= \int_{-\infty}^{\infty} \{R(\mathcal{N})\} p(N_{0,0}, \ldots, N_{r,s}) dN_{0,0} \ldots dN_{r,s} \\
&= \sum_{u,v} G_{-u,-v} \{ \int_{-\infty}^{\infty} N_{u,v} p(N_{u,v}) dN_{u,v} \}
\end{aligned}$$

where we have done some aggressive moving around of variables, and integrated out all the variables that do not appear in each expression in the sum. Since all the

$N_{u,v}$ are independent identically distributed Gaussian random variables with mean μ, we have that:

$$\mathtt{E}[R(\mathcal{N})] = \mu \sum_{u,v} G_{i-u,j-v}$$

The variance of the noise response is obtained as easily. We want to determine

$$\mathtt{E}[\{R(\mathcal{N})_{i,j} - \mathtt{E}[R(\mathcal{N})_{i,j}]\}^2])$$

and this is the same as

$$\int \{\{R(\mathcal{N})_{i,j} - \mathtt{E}[R(\mathcal{N})_{i,j}]\}^2 p(N_{0,0}, \ldots, N_{r,s}) dN_{0,0} \ldots dN_{r,s}$$

which expands to

$$\int \{\sum_{u,v} G_{-u,-v}(N_{u,v} - \mu)\}^2 p(N_{0,0}, \ldots, N_{r,s}) dN_{0,0} \ldots dN_{r,s}$$

This expression expands into a sum of two kinds of integral. Terms of the form

$$\int G^2_{-u,-v}(N_{u,v} - \mu)^2 p(N_{0,0}, \ldots, N_{r,s}) dN_{0,0} \ldots dN_{r,s}$$

(for some u, v) can be integrated easily, because each $N_{u,v}$ is independent; the integral is $\sigma^2 G^2_{-u,-v}$ where σ is the standard deviation of the noise. Terms of the form

$$\int G_{-u,-v} G_{-a,-b}(N_{u,v} - \mu)(N_{a,b} - \mu) p(N_{0,0}, \ldots, N_{r,s}) dN_{0,0} \ldots dN_{r,s}$$

(for some u, v and a, b) integrate to zero, again because each noise term is independent. We now have:

$$\mathtt{E}[\{R(\mathcal{N})_{i,j} - \mathtt{E}[R(\mathcal{N})_{i,j}]\}^2] = \sigma^2 \sum G^2_{u,v}$$

Finite Difference Filters and Gaussian Noise

From these results, we get some insight into the noise behaviour of finite differences. Assume we have an image of stationary Gaussian noise of zero mean, and consider the variance of the response to a finite difference filter that estimates derivatives of increasing order. We shall use the kernel

$$\begin{array}{cc} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{array}$$

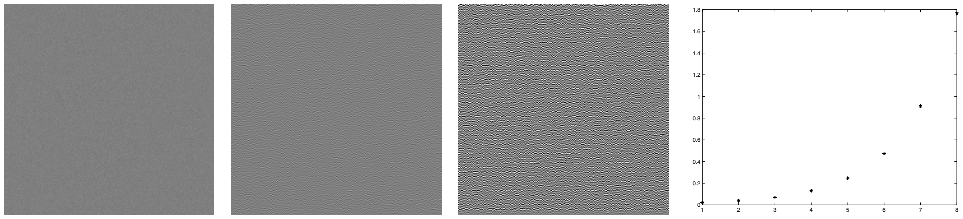


Figure 8.6. Finite differences can accentuate additive Gaussian noise substantially, following the argument in section **??**. On the left, an image of zero mean Gaussian noise with standard deviation 4/256 of the full range. The second figure shows a finite difference estimate of the third derivative in the x direction, and the third shows the sixth derivative in the x direction. In each case, the image has been centered by adding half the full range to show both positive and negative deviations. The images are shown using the same grey level scale; in the case of the sixth derivative, some values exceed the range of this scale. The rightmost image shows the standard deviations of these noise images compared with those predicted by the Pascal's triangle argument.

to estimate the first derivative. Now a second derivative is simply a first derivative applied to a first derivative, so the kernel will be:

$$\begin{array}{ccc} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{array}$$

With a little thought, you can convince yourself that under this scheme, the kernel coefficients of a k'th derivative come from the $k+1$'th row of Pascal's triangle, with appropriate flips of sign. For each of these derivative filters, the mean response to Gaussian noise is zero; but the variance of this response goes up sharply; for the k'th derivative it is the sum of squares of the $k+1$'th row of Pascal's triangle times the standard deviation. Figure 8.6 illustrates this result.

Smoothing alleviates this effect, but the explanation needs a little thought. Assume we smooth a noisy image, and then differentiate it. Firstly, the variance of the noise will tend to be reduced by a smoothing kernel. This is because we tend to use smoothing kernels which are positive, and for which

$$\sum_{uv} G_{uv} = 1$$

which means that

$$\sum_{uv} G_{uv}^2 \leq 1$$

Secondly, pixels will have a greater tendency to look like neighbouring pixels — if we take stationary additive Gaussian noise, and smooth it, the pixel values of the resulting signal *are no longer independent.* In some sense, this is what smoothing

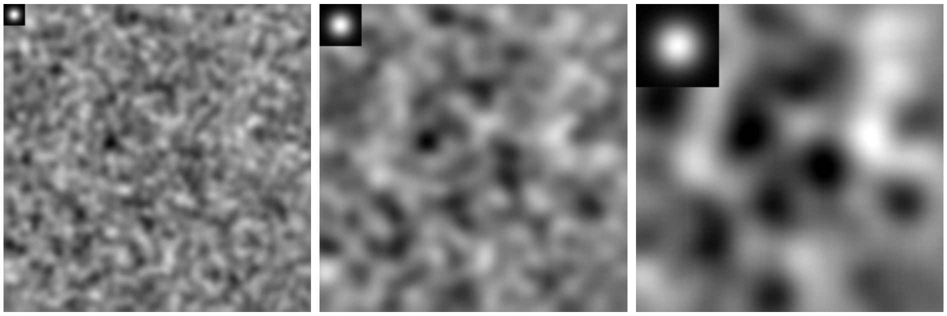


Figure 8.7. Smoothing stationary additive Gaussian noise results in signals where pixel values tend to be increasingly similar to the value of neighbouring pixels. This occurs at about the scale of the filter kernel, because the filter kernel causes the correlations. The figures show noise smoothed with increasingly large Gaussian smoothing kernels. Grey pixels have zero value, darker values are negative and brighter values are positive. The kernels are shown in the top right hand corners of the figures, to indicate the spatial scale of the kernel (we have scaled the brightness of the kernels, which are Gaussians, so that the center pixel is white and the boundary pixels are black). Smoothed noise tends to look like natural texture, as the figures indicate.

was about — recall we introduced smoothing as a method to predict a pixel's value from the values of its neighbours. However, if pixels tend to look like their neighbours, then derivatives must be smaller (because they measure the tendency of pixels to look different from their neighbours).

Smoothed noise has applications. As figure 8.7 indicates, smoothed noise tends to look like some kinds of natural texture, and smoothed noise is quite widely used as a source of textures in computer graphics applications []).

Difficulties with the Additive Stationary Gaussian Noise Model

Taken literally, the additive stationary Gaussian noise model is poor model of image noise. Firstly, the model allows positive (and, more alarmingly, *negative!*) pixel values of arbitrary magnitude. With appropriate choices of standard deviation for typical current cameras operating indoors or in daylight, this doesn't present much of a problem, because these pixel values are extremely unlikely to occur in practice. In rendering noise images, the problematic pixels that do occur are fixed at zero or full output respectively.

Secondly, noise values are completely independent, so this model does not capture the possibility of groups of pixels that have correlated responses, perhaps because of the design of the camera electronics or because of hot spots in the camera integrated circuit. This problem is harder to deal with, because noise models that do model this effect tend to be difficult to deal with analytically. Finally, this model does not describe "dead pixels" (pixels that consistently report no incoming light, or are consistently saturated) terribly well. If the standard deviation is quite large

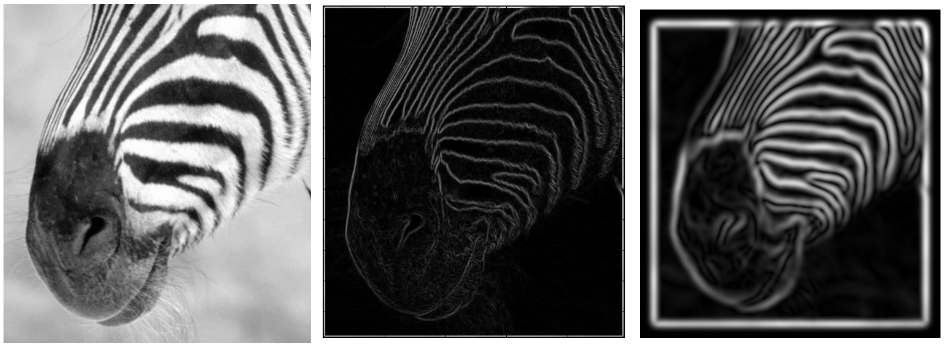


Figure 8.8. The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. On the left, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 1$ pixel and on the right gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 2$ pixel. Notice that large values of the gradient magnitude form thick trails.

and we threshold pixel values, then dead pixels will occur, but the standard deviation may be too large to model the rest of the image well. A crucial advantage of additive Gaussian noise is that it is easy to estimate the response of filters to this noise model. In turn, this gives us some idea of how effective the filter is at responding to signal and ignoring noise.

8.3 Edges and Gradient-based Edge Detectors

Typically, the gradient magnitude can be large along a thick trail in an image (figure 8.8). Object outlines are curves however, and we should like to obtain a curve of the most distinctive points on this trail.

A natural approach is to look for points where the gradient magnitude is a maximum along the direction perpendicular to the edge. For this approach, the direction perpendicular to the edge can be estimated using the direction of the gradient (figure 8.9). These considerations yield algorithm 1. Most current edgefinders follow these lines, but there remain substantial debates about the proper execution of the details.

8.3.1 Estimating Gradients

As figure 8.2 indicates, simple finite difference filters tend to give strong responses to noise, so that applying two finite difference filters is a poor way to estimate a gradient. However, we expect that any change of significance to us has effects over a pool of pixels. For example, the contour of an object can result in a long

```
form an estimate of the image gradient

obtain the gradient magnitude from this estimate

identify image points where the value
of the gradient magnitude is maximal
in the direction perpendicular to the edge
and also large; these points are edge points
```

Algorithm 8.1: *Gradient based edge detection.*

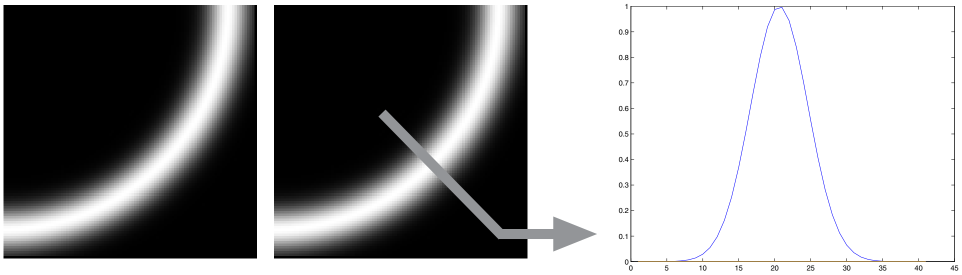


Figure 8.9. The gradient magnitude tends to be large along thick trails in an image. Typically, we would like to condense these trails into curves of representative edge points. A natural way to do this is to cut the trail perpendicular to its direction and look for a peak. We will use the gradient direction as an estimate of the direction in which to cut. The top left figure shows a trail of large gradient magnitude; the figure on the top right shows an appropriate cutting direction; and below, we show the peak in this direction.

chain of points where the image derivative is large. For many kinds of noise model, large image derivatives due to noise are an essentially local event. This means that smoothing a differentiated image would tend to pool support for the changes we are interested in, and to suppress the effects of noise. An alternative interpretation of the point is that the changes we are interested in will not be suppressed by some smoothing, which will tend to suppress the effects of noise. There is no difference in principle between differentiating a smoothed image, or smoothing a differentiated image. In practice, it is usual to differentiate a smoothed image.

8.3.2 Choosing a Smoothing Filter

The smoothing filter can be chosen by taking a model of an edge and then using some set of criteria to choose a filter that gives the best response to that model. It is difficult to pose this problem as a two dimensional problem, because edges in 2D can be curved. Conventionally, the smoothing filter is chosen by formulating a

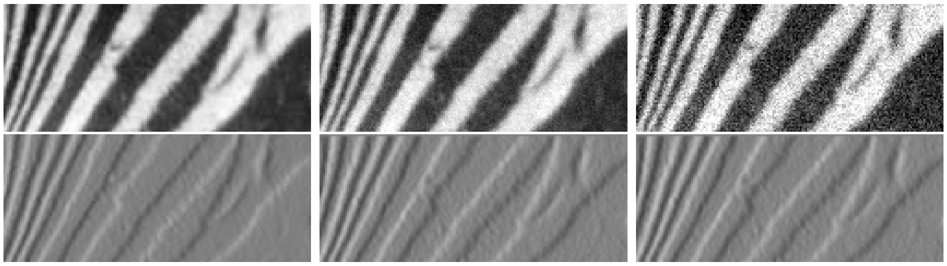


Figure 8.10. Derivative of Gaussian filters are less extroverted in their response to noise than finite difference filters. The image at **top left** shows a detail from a picture of a zebra; **top center** shows the same image corrupted by zero mean stationary additive Gaussian noise, with $\sigma = 0.03$ (pixel values range from 0 to 1). **Top right** shows the same image corrupted by zero mean stationary additive Gaussian noise, with $\sigma = 0.09$. The second row shows the partial derivative in the x-direction of each image, in each case estimated by a derivative of Gaussian filter with σ one pixel. Notice how the smoothing helps to reduce the impact of the noise.

one-dimensional problem, and then using a rotationally symmetric version of the filter in 2D.

The one-dimensional filter must be obtained from a model of an edge. The usual model is a step function of unknown height, in the presence of stationary additive Gaussian noise:

$$edge(x) = AU(x) + n(x)$$

where

$$U(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

(the value of $U(0)$ is irrelevant to our purpose). A is usually referred to as the **contrast** of the edge. In the 1D problem, finding the gradient magnitude is the same as finding the square of the derivative response. For this reason, we usually seek a derivative estimation filter rather than a smoothing filter (which can then be reconstructed by integrating the derivative estimation filter).

Canny established the practice of choosing a derivative estimation filter by using the continuous model to optimize a combination of three criteria:

- **Signal to noise ratio** — the filter should respond more strongly to the edge *at* $x = 0$ than to noise.
- **Localisation** — the filter response should reach a maximum very close to $x = 0$.
- **Low false positives** — there should be only one maximum of the response in a reasonable neighbourhood of $x = 0$.

Once a continuous filter has been found, it is discretised. The criteria can be combined in a variety of ways, yielding a variety of somewhat different filters. It is a remarkable fact that the optimal smoothing filters that are derived by most combinations of these criteria tend to look a great deal like Gaussians — this is intuitively reasonable, as the smoothing filter must place strong weight on center pixels and less weight on distant pixels, rather like a Gaussian. In practice, optimal smoothing filters are usually replaced by a Gaussian, with no particularly important degradation in performance.

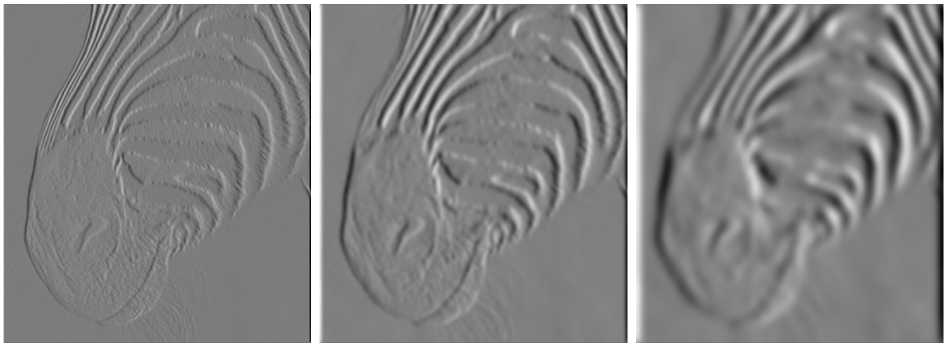


Figure 8.11. The scale (i.e. σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra, obtained using a derivative of Gaussian filter with σ one pixel, three pixels and seven pixels (moving to the right). Note how images at a finer scale show some hair and the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.

The choice of σ used in estimating the derivative is often called the *scale* of the smoothing. Scale has a substantial effect on the response of a derivative filter. Assume we have a narrow bar on a constant background, rather like the zebra's whisker. Smoothing on a scale smaller than the width of the bar will mean that the filter responds on each side of the bar, and we will be able to resolve the rising and falling edges of the bar. If the filter width is much greater, the bar will be smoothed into the background, and the bar will generate little or no response (as figure 8.11).

8.3.3 Why Smooth with a Gaussian?

While a Gaussian is not the only possible blurring kernel, it is convenient because it has a number of important properties. Firstly, if we convolve a Gaussian with a Gaussian, and the result is another Gaussian:

$$G_{\sigma_1} * *G_{\sigma_2} = G_{\sqrt{\sigma_1^2+\sigma_2^2}}$$

This means that it is possible to obtain very heavily smoothed images by resmoothing smoothed images. This is a significant property, firstly because discrete convolution can be an expensive operation (particularly if the kernel of the filter is large), and secondly because it is common to want to see versions of an image smoothed by different amounts.

Efficiency

Consider convolving an image with a Gaussian kernel with σ one pixel. Although the Gaussian kernel is non zero over an infinite domain, for most of that domain it is extremely small because of the exponential form. For σ one pixel, points outside a 5x5 integer grid centered at the origin have values less than $e^{-4} = 0.0184$ and points outside a 7x7 integer grid centered at the origin have values less than $e^{-9} = 0.0001234$. This means that we can ignore their contributions, and represent the discrete Gaussian as a small array (5x5 or 7x7, according to taste and the number of bits you allocate to representing the kernel).

However, if σ is 10 pixels, we may need a 50x50 array or worse. A back of the envelope count of operations should convince you that convolving a reasonably sized image with a 50x50 array is an unattractice prospect. The alternative — convolving repeatedly with a much smaller kernel — is much more efficient, *because we don't need to keep every pixel in the interim.* This is because a smoothed image is, to some extent, redundant (most pixels contain a significant component of their neighbours' values). As a result, some pixels can be discarded. We then have a strategy which is quite efficient: smooth, subsample, smooth, subsample, etc. The result is an image that has the same information as a heavily smoothed image, but is very much smaller and is easier to obtain. We explore the details of this approach in section 7.5.1.

The Central Limit Theorem

Gaussians have another significant property which we shall not prove but illustrate in figure 8.12. For an important family of functions, convolving any member of that family of functions with itself repeatedly will eventually yield a Gaussian. With the associativity of convolution, this implies that if we choose a different smoothing kernel, and apply it repeatedly to the image, the result will eventually look as though we had smoothed the image with a Gaussian anyhow.

Gaussians are Separable

Finally, a Gaussian can be factored as

$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2 + y^2)}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x^2)}{2\sigma^2}\right)\right) \times \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^2)}{2\sigma^2}\right)\right) \end{aligned}$$

and this is a product of two 1-D Gaussians. Generally, a function $f(x, y)$ that factors as $f(x, y) = g(x)h(y)$ is referred to as a **tensor product**. It is common to

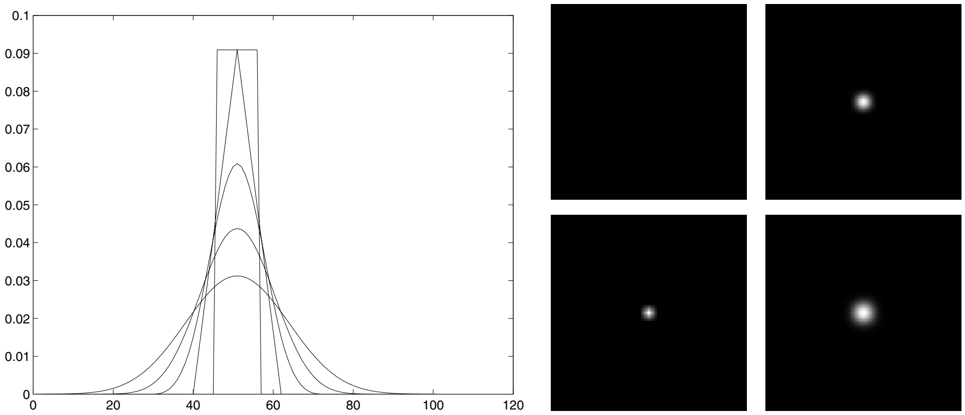


Figure 8.12. The central limit theorem states that repeated convolution of a positive kernel with itself will eventually limit towards a kernel that is a scaling of a Gaussian. The top figure illustrates this effect for 1D convolution; the triangle is obtained by convolving a box function with itself; each succeding stage is obtained by convolving the previous stage with itself. The four images show a 2D box function convolved with itself 0, 1, 2 and 4 times (clockwise).

refer to filter kernels that are tensor products as **separable kernels**. Separability is a very useful property indeed. In particular, convolving with a filter kernel that is separable is the same as convolving with two 1-D kernels, one in the x direction and another in the y direction (exercises).

Many other kernels are separable. Separable filter kernels result in discrete representations that factor as well. In particular, if $\mathcal{H}$ is a discretised separable filter kernel, then there are some vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ such that

$$H_{ij} = f_i g_j$$

It is possible to identify this property using techniques from numerical linear algebra; commercial convolution packages often test the kernel to see if it is separable before applying it to the image. The cost of this test is easily paid off by the savings if the kernel does turn out to be separable.

8.3.4 Derivative of Gaussian Filters

Smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. This fact is most easily seen by thinking about continuous convolution.

Firstly, differentiation is linear and shift invariant. This means that there is some kernel — we dodge the question of what it looks like — that differentiates.

That is, given a function $I(x, y)$

$$\frac{\partial I}{\partial x} = K_{\frac{\partial}{\partial x}} * *I$$

Now we want the derivative of a smoothed function. We write the convolution kernel for the smoothing as S. Recalling that convolution is associative, we have

$$(K_{\frac{\partial}{\partial x}} * *(S * *I)) = (K_{\frac{\partial}{\partial x}} * *S) * *I = (\frac{\partial S}{\partial x}) * *I$$

This fact appears in its most commonly used form when the smoothing function is a Gaussian; we can then write

$$\frac{\partial\, (G_\sigma * *I)}{\partial x} = (\frac{\partial G_\sigma}{\partial x}) * *I$$

i.e. we need only convolve with the derivative of the Gaussian, rather than convolve and then differentiate. A similar remark applies to the Laplacian. Recall that the Laplacian of a function in 2D is defined as:

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Again, because convolution is associative, we have that

$$(K_{\nabla^2} * *(G_\sigma * *I)) = (K_{\nabla^2} * *G_\sigma) * *I = (\nabla^2 G_\sigma) * *I$$

This practice results in much smaller noise responses from the derivative estimates (figure 8.10).

8.3.5 Identifying Edge Points from Filter Outputs

Given estimates of gradient magnitude, we would like to obtain edge points. Again, there is clearly no objective definition, and we proceed by reasonable intuition. The gradient magnitude can be thought of as a chain of low hills. Marking local extrema would mark isolated points — the hilltops in the analogy. A better criterion is to slice the gradient magnitude along the gradient direction — which should be perpendicular to the edge — and mark the points along the slice where the magnitude is maximal. This would get a chain of points along the crown of the hills in our chain; the process is called **non-maximum suppression**.

Typically, we expect edge points to occur along curve-like chains. The significant steps in non maximum suppression are:

- determining whether a given point is an edge point;
- and, if it is, finding the next edge point.

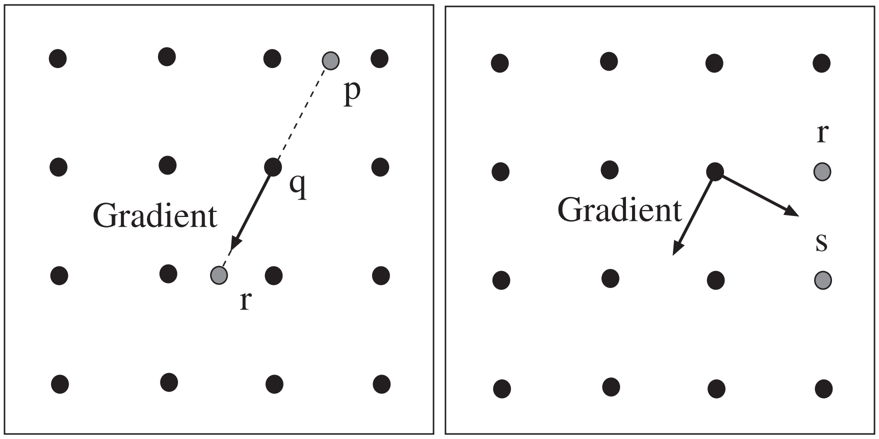


Figure 8.13. Non-maximum suppression obtains points where the gradient magnitude is at a maximum *along the direction of the gradient.* The figure on the left shows how we reconstruct the gradient magnitude. The dots are the pixel grid. We are at pixel $\boldsymbol{q}$, attempting to determine whether the gradient is at a maximum; the gradient direction through $\boldsymbol{q}$ does not pass through any convenient pixels in the forward or backward direction, so we must interpolate to obtain the values of the gradient magnitude at $\boldsymbol{p}$ and $\boldsymbol{r}$; if the value at $\boldsymbol{q}$ is larger than both, $\boldsymbol{q}$ is an edge point. Typically, the magnitude values are reconstructed with a linear interpolate, which in this case would use the pixels to the left and right of $\boldsymbol{p}$ and $\boldsymbol{r}$ respectively to interpolate values at those points. On the right, we sketch how to find candidates for the next edge point, given that $\boldsymbol{q}$ is an edge point; an appropriate search direction is perpendicular to the gradient, so that points $\boldsymbol{s}$ and $\boldsymbol{t}$ should be considered for the next edge point. Notice that, in principle, we don't need to restrict ourselves to pixel points on the image grid, because we know where the predicted position lies between $\boldsymbol{s}$ and $\boldsymbol{t}$, so that we could again interpolate to obtain gradient values for points off the grid.

Once these steps are understood, it is easy to enumerate all edge chains. We find the first edge point, mark it, expand all chains through that point exhaustively, marking all points along those chains, and continue to do this for all unmarked edge points.

The two main steps are simple. For the moment, assume that edges are to be marked at pixel locations (rather than, say, at some finer subdivision of the pixel grid). We can determine whether the gradient magnitude is maximal at any pixel by comparing it with values at points some way backwards and forwards *along the gradient direction.* This is a function of distance along the gradient; typically we step forward to the next row (or column) of pixels and backwards to the previous to determine whether the magnitude at our pixel is larger (figure 8.13). The gradient direction does not usually pass through the next pixel, so we must interpolate to determine the value of the gradient magnitude at the points we are interested in; a linear interpolate is usual.

```
While there are points with high gradient
that have not been visited

  Find a start point that is a local maximum in the
    direction perpendicular to the gradient
    erasing points that have been checked

  while possible, expand a chain through
    the current point by:
      1) predicting a set of next points, using
        the direction perpendicular to the gradient

      2) finding which (if any) is a local maximum
        in the gradient direction

      3) testing if the gradient magnitude at the
        maximum is sufficiently large

      4) leaving a record that the point and
        neighbours have been visited

      record the next point, which becomes the current point

  end

end
```

Algorithm 8.2: *Non-maximum suppression.*

If the pixel turns out to be an edge point, the next edge point in the curve can be guessed by taking a step perpendicular to the gradient. This step will not, in general, end on a pixel; a natural strategy is to look at the neighbouring pixels that lie close to that direction (see figure 8.13. This approach leads to a set of curves that can be represented by rendering them in black on a white background, as in figure **??**.

There are too many of these curves to come close to being a reasonable representation of object boundaries. This is, in part, because we have marked maxima of the gradient magnitude without regard to how large these maxima are. It is more usual to apply a threshold test, to ensure that the maxima are greater than some lower bound. This in turn leads to broken edge curves (figure **??**). The usual trick

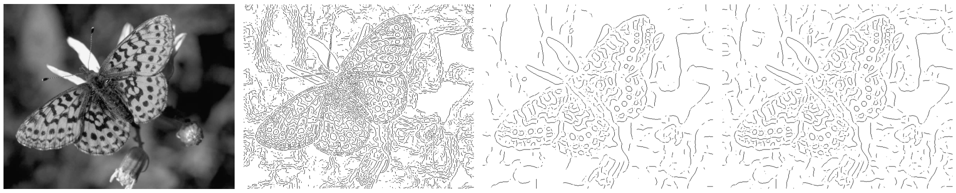


Figure 8.14. Edge points marked on the pixel grid for the image shown on the **left**. The edge points on the **center left** are obtained using a Gaussian smoothing filter at σ one pixel and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **center right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a low threshold to determine whether a point is an edge point or not. At a fine scale, fine detail at high contrast generates edge points, which disappear at the coarser scale. When the threshold is high, curves of edge points are often broken because the gradient magnitude dips below the threshold; for the low threshold, a variety of new edge points of dubious significance are introduced.

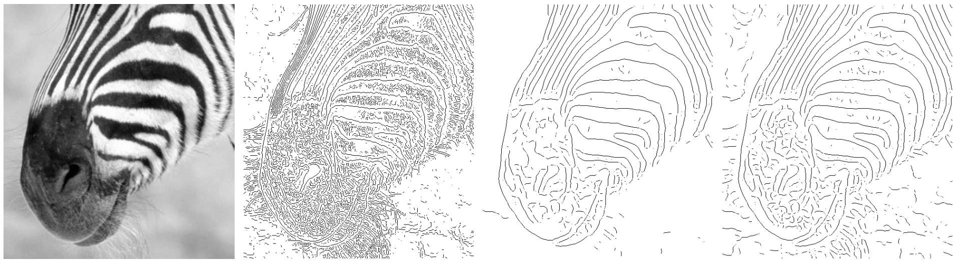


Figure 8.15. Edge points marked on the pixel grid for the image shown on the **left**. The edge points on the **center left** are obtained using a Gaussian smoothing filter at σ one pixel and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **center right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a low threshold to determine whether a point is an edge point or not. At a fine scale, fine detail at high contrast generates edge points, which disappear at the coarser scale. When the threshold is high, curves of edge points are often broken because the gradient magnitude dips below the threshold; for the low threshold, a variety of new edge points of dubious significance are introduced.

for dealing with this is to use **hysteresis**; we have two thresholds, and refer to the *larger* when starting an edge chain and the *smaller* while following it. The trick often results in an improvement in edge outputs (exercises)

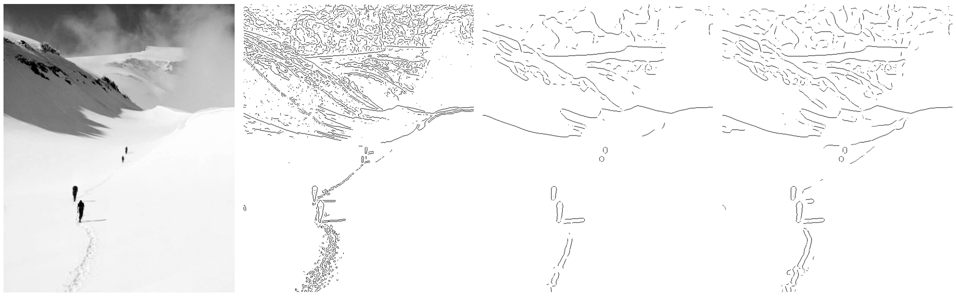


Figure 8.16. Edge points marked on the pixel grid for the image shown on the **left**. The edge points on the **center left** are obtained using a Gaussian smoothing filter at σ one pixel and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **center right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point or not. The edge points on the **right** are obtained using a Gaussian smoothing filter at σ four pixels and gradient magnitude has been tested against a low threshold to determine whether a point is an edge point or not. At a fine scale, fine detail at high contrast generates edge points, which disappear at the coarser scale. When the threshold is high, curves of edge points are often broken because the gradient magnitude dips below the threshold; for the low threshold, a variety of new edge points of dubious significance are introduced.

8.4 Commentary

Edge detection is a subject that is alive with controversy, much of it probably empty. We have hardly scratched the surface. There are many optimality criteria for edge detectors, and rather more "optimal" edge detectors. At the end of the day, most boil down to smoothing the image with something that looks a lot like a Gaussian before measuring the gradient. Our choice of Canny's approach to expound here and in the next chapter is based mainly on tradition.

Object boundaries are not the same as sharp changes in image values. Firstly, objects may not have a strong contrast with their backgrounds through sheer bad luck. Secondly, objects are often covered with texture or markings which generate edges of their own; often so many that it is hard to wade through them to find the relevant pieces of object boundary. Finally, shadows and the like may generate edges that have no relation to object boundaries. There are some strategies for dealing with these difficulties.

Firstly, some applications allow management of illumination; if it is possible to choose the illumination, a careful choice can make a tremendous difference in the contrast and eliminate shadows. Secondly, by setting smoothing parameters large and contrast thresholds high it is often possible to ensure that edges due to texture are smoothed over and not marked. This is a dubious business, firstly because it can be hard to choose reliable values of the smoothing and the thresholds and secondly because it is perverse to regard texture purely as a nuisance, rather than a source of information.

There are other ways to handle the uncomfortable distinction between edges and object boundaries. Firstly, one might work to make better edge detectors. This approach is the root of a huge literature, dealing with matters like localisation, corner topology and the like. We incline to the view that returns are diminishing rather sharply in this endeavour.

Secondly, one might deny the usefulness of edge detection entirely. This approach is rooted in the observation that some stages of edge detection, particularly non-maximum suppression, discard information that is awfully difficult to retrieve later on. This is because a hard decision — testing against a threshold — has been made. Instead, the argument proceeds, one should keep this information around in a "soft" (a propaganda term for probabilistic) way. Attactive as these arguments sound, we are inclined to discount this view because there are currently no practical mechanisms for handling the volumes of soft information so obtained.

Finally, one might regard this as an issue to be dealt with by overall questions of system architecture — the fatalist view that almost every visual process is going to have obnoxious features, and the correct approach to this problem is to understand the integration of visual information well enough to construct vision systems that are tolerant to this. Although it sweeps a great deal of dust under the carpet — precisely *how* does one construct such architectures? — we find this approach most attractive and will discuss it again and again.

All edge detectors behave badly at corners; only the details vary. This has resulted in two lively strands in the literature (i - what goes wrong; ii - what to do about it). There are a variety of quite sophisticated corner detectors, mainly because corners make quite good point features for correspondence algorithms supporting such activities as stereopsis, reconstruction or structure from motion. This has led to quite detailed practical knowledge of the localisation properties of corner detectors (e.g. []).

A variety of other forms of edge are quite common, however. The most commonly cited example is the **roof edge**, which can result from the effects of interreflections (figure 2.17). Another example that also results from interreflections is a composite of a step and a roof (figure **??**). It is possible to find these phenomena by using essentially the same steps as outlined above (find an "optimal" filter, and do non-maximum suppression on its outputs). In practice, this is seldom done. There appear to be two reasons. Firstly, there is no comfortable basis in theory (or practice) for the models that are adopted — what particular composite edges are worth looking for? the current answer — those for which optimal filters are reasonably easy to derive — is most unsatisfactory. Secondly, the semantics of roof edges and more complex composite edges is even vaguer than that of step edges — there is little notion of what one would *do* with roof edge once it had been found.

Edges are poorly defined and usually hard to detect, but one can solve problems with the output of an edge detector. Roof edges are similarly poorly defined and similarly hard to detect; we have never seen problems solved with the output of a roof edge detector. The real difficulty here is that there seems to be no reliable mechanism for predicting, in advance, what will be worth detecting. We will scratch the surface of this very difficult problem in section **??**.

Assignments

Exercises

1. We said "One diagnostic for a large gradient magnitude is a zero of a "second derivative" at a point where the gradient is large. A sensible 2D analogue to the 1D second derivative must be rotationally invariant" in section 8.1.2. Why is this true?

Programming Assignments

1. Why is it necessary to check that the gradient magnitude is large at zero crossings of the Laplacian of an image? Demonstrate a series of edges for which this test is significant.

2. The Laplacian of a Gaussian looks similar to the difference between two Gaussians at different scales. Compare these two kernels for various values of the two scales — which choices give a good approximation? How significant is the approximation error in edge finding using a zero-crossing approach?

3. Obtain an implementation of Canny's edge detector (you could try the vision home page at `http://www.somewhereorother`) and make a series of images indicating the effects of scale and contrast thresholds on the edges that are detected. How easy is it to set up the edge detector to mark only object boundaries? Can you think of applications where this would be easy?

4. It is quite easy to defeat hysteresis in edge detectors that implement it — essentially, one sets the lower and higher thresholds to have the same value. Use this trick to compare the behaviour of an edge detector with and without hysteresis. There are a variety of issues to look at:

    - What are you trying to do with the edge detector output? it is sometimes very helpful to have linked chains of edge points — does hysteresis help significantly here?
    - Noise suppression: we often wish to force edge detectors to ignore some edge points and mark others. One diagnostic that an edge is useful is high contrast (it is by no means reliable). How reliably can you use hysteresis to suppress low contrast edges without breaking high contrast edges?

Chapter 9

FILTERS AND FEATURES

Linear filters can be thought of as simple pattern finders. This view is helpful, because it allows us to build quite effective object detection systems using filters. We describe this view of a filter in section 9.1; in section 9.2, we use this view to understand a body of evidence about the primate early vision system, and in section 9.3 we describe a system that uses this approach to find hand gestures. Section 9.4 shows how to find other kinds of features — in particular, corners — using filter outputs. Finally, we describe more complex noise models and correspondingly more complex smoothing techniques in section 9.5.

9.1 Filters as Templates

It turns out that filters offer a natural mechanism for finding simple patterns, because filters respond most strongly to pattern elements that look like the filter. For example, smoothed derivative filters are intended to give a strong response at a point where the derivative is large; at these points, the kernel of the filter "looks like" the effect it is intended to detect. The x-derivative filters look like a vertical light blob next to a vertical dark blob (an arrangement where there is a large x-derivative), and so on.

It is generally the case that filters that are intended to give a strong response to a pattern look like that pattern. This is a simple geometric result.

9.1.1 Convolution as a Dot Product

Recall from section 7.1.1 that, for $\mathcal{G}$ the kernel of some linear filter, the response of this filter to an image $\mathcal{H}$ is given by:

$$R_{ij} = \sum_{u,v} G_{i-u,j-v} H_{uv}$$

Now consider the response of a filter at the point where i and j are zero. This will be

$$R = \sum_{u,v} G_{-u,-v} H_{u,v}$$

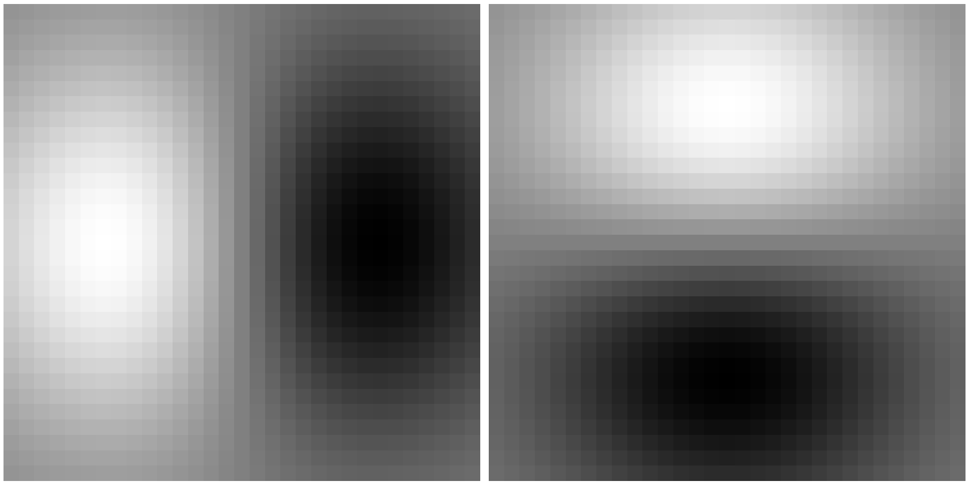


Figure 9.1. Filter kernels "look like" the effects they are intended to detect. On the left, an smoothed derivative of Gaussian filter that looks for large changes in the x-direction (such as a dark blob next to a light blob); on the right, a smoothed derivative of Gaussian filter that looks for large changes in the y-direction.

This response is obtained by associating image elements with filter kernel elements, multiplying the associated elements, and summing. We could scan the image into a vector, and the filter kernel into another vector, in such a way that associated elements are in the same component. By inserting zeros as needed, we can ensure that these two vectors have the same dimension. Once this is done, the process of multiplying associated elements and summing is precisely the same as taking a dot-product.

This is a powerful analogy, because this dot-product, like any other, will achieve its largest value when the vector representing the image is parallel to the vector representing the filter kernel. This means that a filter responds most strongly when it encounters an image pattern that looks like the filter. The response of a filter will get stronger as a region gets brighter, too.

Now consider the response of the image to a filter at some other point. Nothing significant about our model has changed; again, we can scan the image into one vector and the filter kernel into another vector, such that associated elements lie in the same components. Again, the result of applying this filter is a dot-product. There are two useful ways to think about this dot-product.

9.1.2 Changing Basis

We can think of convolution as a dot-product between the image and *a different vector* (because we have moved the filter kernel to lie over some other point in the image). The new vector is obtained by rearranging the old one, so that the elements lie in the right components to make the sum work out (exercise **??**). This means that, by convolving an image with a filter, we are representating the image on a new *basis* of the vector space of images — the basis given by the different shifted versions of the filter. The original basis elements were vectors with a zero in all slots except one. The new basis elements are shifted versions of a single pattern. For many of the kernels we have discussed, we expect that this process will *lose* information — for the same reason that smoothing suppresses noise — so that the coefficients on this basis are redundant. This basis transformation is valuable in texture analysis (section **??**).

9.2 Human Vision: Filters and Primate Early Vision

There is quite a lot of information about the early stages of the primate visual system. The "wiring" can be studied using stains that carry from cell to cell; the response of individual cells can be studied by displaying patterns and recording the electrical behaviour of the cell; and some structural information can be elicited using psychophysical experiments. All this evidence suggests that spatiotemporal filters yield a rather good model of early visual processing.

9.2.1 The Visual Pathway

The anatomical details of how visual information is passed into the brain, and what happens in the early stages, are quite well understood. Information about the connections along this pathway can be obtained by staining methods; typically, a cell is stained with a substance that moves in known ways (along the body of the cell; across connections; etc.) and one looks to see where the stain ends up.

The stages in the **visual pathway** are:

- The **retina**, where photoreceptive cells transduce irradiance to electrical spikes. These signals are processed by a variety of layers of cells. **Retinal ganglion cells** connect to the final layer.
- The **optic nerve** consists of the fibers of the retinal ganglion cells, and connects the retina to the brain through the the **optic chiasma**. This is a crossing point; the left-hand side of *each* retina is connected to the left half of the brain, and the right-hand side to the right half.
- There are now two pathways; some information is fed to the **superior colliculus** (which we shall ignore), but most goes to the **lateral geniculate nucleus**.
- The lateral geniculate nucleus is connected to the **visual cortex**, one of the best studied regions in the primate brain. The visual cortex consists of a series of quite well defined layers. Much of early vision occurs in this structure, which contains a large selection of different representations of an image, organised in fairly well understood structures.

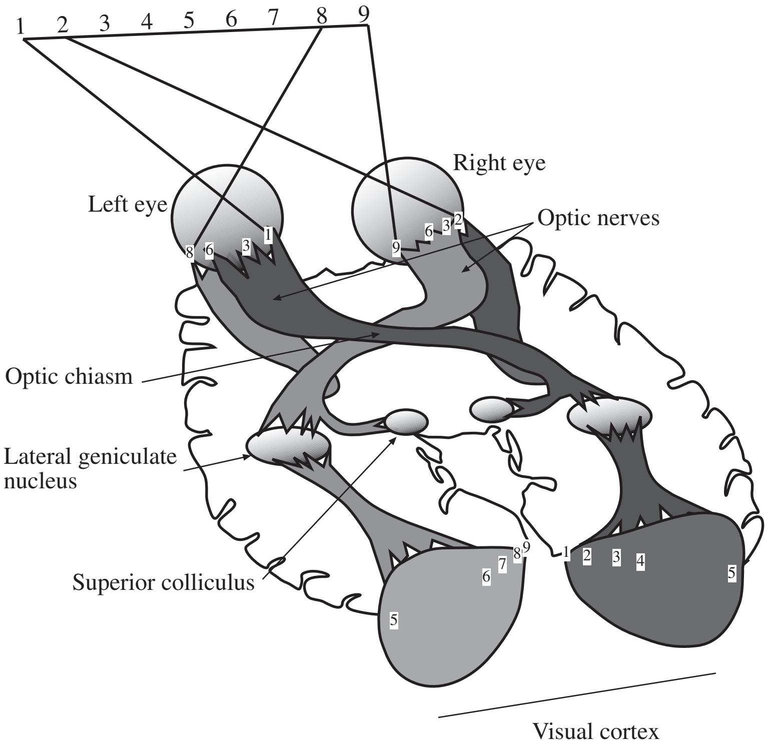

1 2 3 4 5 6 7 8 9
Right eye
Left eye
8 6 3 1
9 6 3 2
Optic nerves
Optic chiasm
Lateral geniculate nucleus
Superior colliculus
5 6 7 8 9
1 2 3 4 5
Visual cortex

Figure 9.2. A diagram of the visual pathway, viewed from the underside of the brain. Elements of the right hand visual field are imaged on both left and right retinas; through the optic chiasm, the left hand visual field is projected to the right lateral geniculate nucleus (and vice versa — the dashed lines indicate the pathway followed by the right visual field). Signals then leave the LGN for the visual cortex, at the back of the brain. The behaviour of cells up to and including the cortex is quite well understood, and is usually modelled as a system of filters. Notice that the outputs of retinal cells map to the cortex in a spatially organised fashion, but that the central portion of the visual field maps to a larger area of cortex than the peripheral portions; this reflects the fact that the eye has higher spatial resolution in the central portion of the visual field. Redrawn after figure ****** of Frisby.

- Visual information leaves the visual cortex for the **parietal cortex** and the **temporal cortex**. The temporal cortex appears to be involved in determining *what* an object is, and the parietal cortex in determining *where* it is [] and possibly *how* it can be grasped []. This information must be reunited somewhere, but it isn't currently known where.

9.2.2 How the Visual Pathway is Studied

There are quite detailed models of how cells respond along this pathway, up to the point where signals leave the visual cortex (there is rather little information after this point). These models are supported by large quantities of evidence on the responses elicited from cells in the brain by particular patterns of light and motion. Both the visual cortex and the lateral geniculate nucleus are quite conveniently sited on a primate brain, so that it is possible to insert electrodes into these structures without traumatising the brain so badly that the results are meaningless.

In a typical experiment, an electrode is inserted along a path through some structure to record electrical signals from cells along that path. Patterns of light, colour and motion are shown to an experimental animal, and the response from the electrode — which is hopefully a response from a nearby cell — is recorded along with the depth to which the electrode has penetrated. When sufficient information has been obtained, the electrode is moved deeper along its path and the process continues. Recordings may be taken from several different paths. Eventually, the experimental animal is sacrificed and its brain cut in sections to determine from what precise regions the electrode recorded information.

Gratings

Typically, neurons in the visual pathway are discussed in terms of their **receptive field** — this is a record of the spatial distribution of the effect of illumination on the neuron's output. The response of cells is often determined by their response to a grating. A **spatial grating** is a pattern of the form $m(1+\cos 2\pi f x)$, where x is a convenient spatial coordinate across the visual field. These gratings can be used to investigate the spatial response of a cell; the spatio-temporal response is studied using a **spatio-temporal grating** is a pattern of the form $m(1+a\cos 2\pi f x \cos 2\pi g t))$ — this is a moving sinusoid. Many cells temporal components to their response as well, and such cells can be described in terms of their contrast sensitivity to a spatio-temporal grating (figure 9.3).

9.2.3 The Response of Retinal Cells

The earliest cells in the visual pathway are **retinal ganglion cells**, which collect outputs from the layers of retinal cells receptors. Typically, light falling in the center of a ganglion cell's receptive field increases its firing rate — often called its response — and light falling in the outside decreases the firing rate. A cell that responds like this is referred to as an **on-center, off-surround** cell; there are also **off-center, on-surround** cells. *For a fixed mean value m*, ganglion cells appear to sum their response over the receptive field, weighting the center positive (respectively negative) and the surround negative (resp. positive) for an on-center, off-surround (resp. off-center, on-surround) cell. In particular, their response appears to be linear. This can be tested by comparing the sum of the responses to individual stimuli and the response to a sum of these stimuli. Figure 9.3 illustrates the behaviour of a linear retinal ganglion cell.

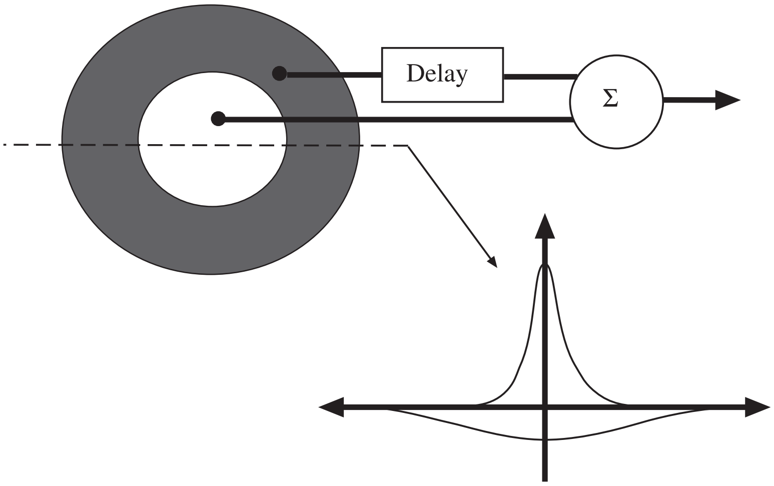


Figure 9.3. The response of a retinal ganglion cell can be predicted by adding the temporal response of the center to the temporal response of the surround (which is slower than that of the center, and is so delayed in the model). As the cross-section indicates, the model of the spatial response is a difference of Gaussian model — there is a center field that has a spatial sensitivity of the form of a narrow Gaussian, and a surround field that has the form of a broad Gaussian. One field excites, the other inhibits (as indicated by the positive/negative signs in the figure). Figure redrawn fter Enroth-Cugell et al.,

This linearity can be exploited to measure response using periodic functions, which are usually spatial sinusoids. It is usual to study neurons by fixing some level of response, and then determining the **contrast** (amplitude of the sinusoid) required to elicit that level of response — the **contrast threshold**. Typically, one plots the **contrast sensitivity** (the inverse of the contrast threshold) against some interesting variable. Figure 9.4 plots the contrast sensitivity of a center-surround neuron against a measure of spatial frequency. In this case, the stimulus shown was fixed, and the cell's response measured after some long time, giving an assymptotic response. The contrast sensitivity function is different for different mean values, an effect known as **adaptation**.

9.2.4 The Lateral Geniculate Nucleus

The LGN is a layered structure, consisting of many sheets of neurons. The layers are divided into two classes — those consisting of cells with large cell bodies (**magnocellular layers**), and those consisting of cells with small cell bodies (**parvocellular layers**). The behaviour of these cells differs as well.

Each layer in the LGN receives input from a single eye, and is laid out like the retina of the eye providing input (an effect known as **retinotopic mapping**).

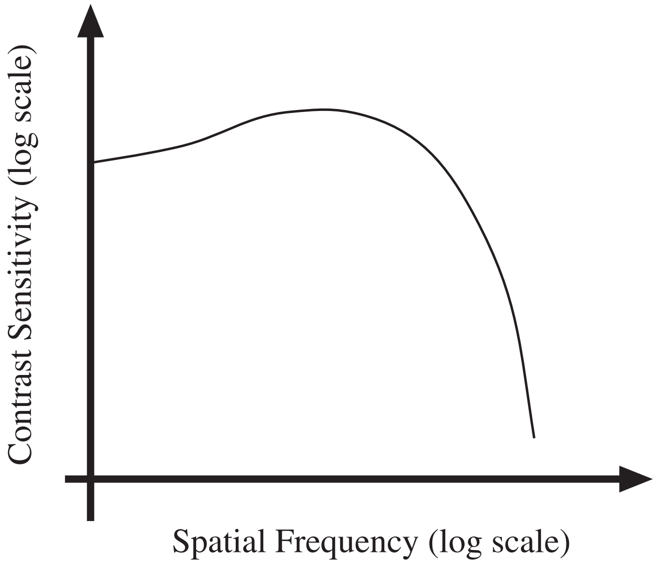


Figure 9.4. The contrast sensitivity function can be used to describe the behaviour of linear cells. In a center-surround cell, described in the figure, the response to a constant stimulus is zero. The receptive field is symmetric, meaning that its behaviour can be described in terms of its response to a signal of the form $m(1 + a\cos(2\pi f x))$, where f is the spatial frequency and x is spatial position in a convenient system of units; a is referred to the contrast of the signal. The contrast sensitivity is obtained by fixing some level of response, and taking the inverse of the contrast required to reach that level of response. Center-surround cells are tuned in spatial frequency; if the spatial frequency of the signal is low, then the signal is nearly constant over the receptive field of the cell, and the contrast sensitivity is lowered. If the frequency of the signal is high, then the excitatory and inhibitory responses tend to cancel. These considerations lead to a curve with the rather typical shape shown here.

Retinotopic mapping means that nearby regions on the retina end up near one another in the layer, and so we can think of each layer as representing some form of feature map. Neurons in the lateral geniculate nucleus display similar receptive field behaviour to retinal neurons. The role of the LGN is unclear; it is known that the LGN receives input from the cortex and from other regions of the brain, which may modify the visual signal travelling from the retina to the cortex.

9.2.5 The Visual Cortex

Most visual signals arrive at an area of the cortex called **area V1** (or the **primary visual cortex**, or the **striate cortex**). This area is highly structured and has been intensively studied. Most physiological information about the cortex comes from studies of cats or monkeys (which are known to react differently from one another and from humans if the cortex is damaged). The cortex is also retinotopically mapped. It has a highly structured layered architecture, with regions organised by the eye of origin of the signal (often called **ocular dominance columns**). Within these columns, cells are arranged so that their receptive fields move smoothly from the center to the periphery of the visual field. Neurons in the primary visual cortex have been extensively studied. Two classes are usually recognised — **simple cells** and **complex cells**.

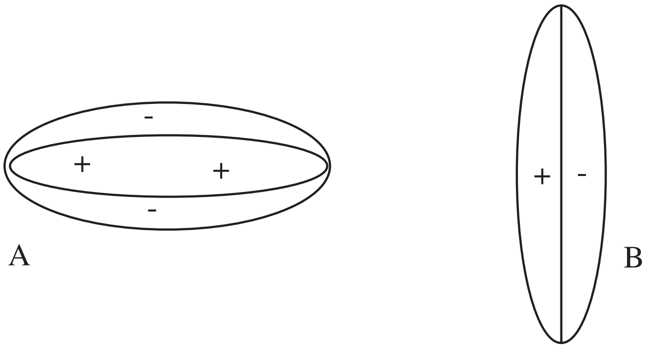


Figure 9.5. Cortical simple cells can typically be modelled as linear, and are usually thought of as spot, edge or bar detectors. Typical receptive fields are shown above; the output of the cell on the left is inhibited by light in the center of its receptive field — marked with the '-' sign — and excited by light in the surround (the '+' sign). Thus, the cell on the left is particularly strongly excited by dark bars on a light background, and so is a bar detector. Similarly, the cell on the right is strongly excited by a vertical edge (a bright patch next to a dark patch). Notice that these cells are orientation selective; the cell on the left responds most strongly to a bright bar on a dark background and the cell on the right will respond strongly to an oriented edge. Notice the similarity to the derivative of Gaussian filters plotted below, where the mid-grey level represents a zero value in the kernel, a dark value is negative and a light value is positive.

Simple cells have **orientation selective** receptive fields, meaning that a particular cell will respond more strongly to an oriented structure. To a good approximation, the response of a simple cell is linear, so that the behaviour of these cells can be modelled with spatial filters. The structure of typical receptive fields means that these cells can be thought of as edge and bar detectors [**?**], or as first and second derivative operators. Some simple cells have more lobes to their receptive field, and can be thought of as detecting higher derivatives. The preferred orientation of a cell varies fairly smoothly in a principled way that depends on the cell's position.

Complex cells typically are highly non-linear, and respond to moving edges or bars. Typically, the cells display **direction selectivity**, in that the response of a cell to a moving bar depends strongly on both the orientation of the bar and the direction of the motion (figure 9.6). Some complex cells, often called **hypercomplex cells** respond preferentially to bars of a particular length.

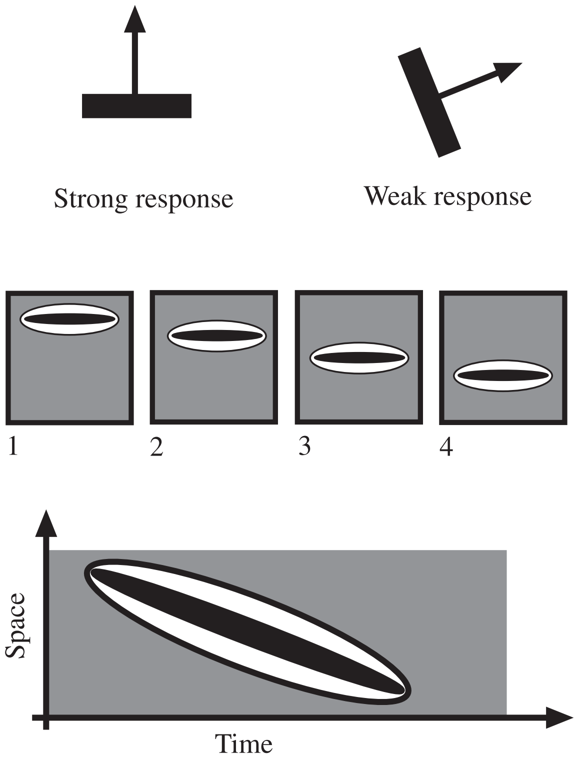


Figure 9.6. Many cortical cells typically give stronger or weaker responses to moving bars, depending on the direction in which the bar moves. The receptive field of a cell like this can be thought of as a spatial filter that is swept in a particular direction with time — a **spatio-temporal filter**. The first row shows typical behaviour from such a cell; a horizontal bar moving vertically gets a strong response, but the bar gets a much weaker response when its orientation changes. The second row shows one way of thinking about this phenomenon — the cell's output is computed by adding together the response of several different spatial filters, computed at different time offsets. An alternative way to think of this cell is as a linear filter in the spatial and the temporal domain; we can lay out the kernel as a graph in space and time, as on the bottom row.

One strong distinction between simple and complex cells appears when one considers the time-course of a response to a **contrast reversing pattern** — a spatial sinusoid whose amplitude is a sinusoidal function of time. Exposed to such a stimulus, a simple cell responds strongly for the positive contrast and not at all for the negative contrast — it is trying to be linear, but because the resting response of the cell is low, there is a limit to the extent to which the cell's output can be inhibited. In contrast, complex cells respond to both phases (figure 9.7). Thus, one can think of a simple cell as performing half-wave rectification — it responds to

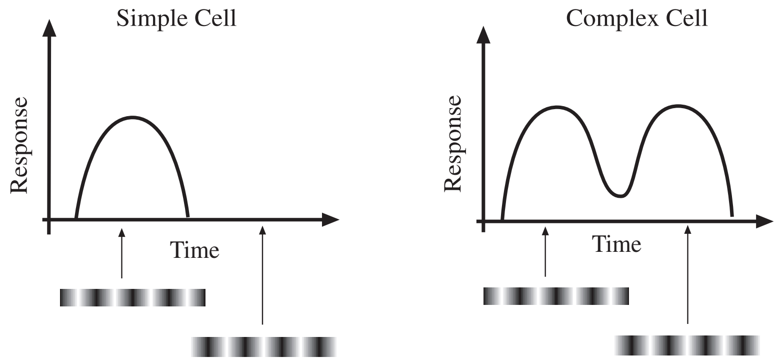


Figure 9.7. Cortical cells are classified simple or complex, depending on their response to a time-reversing grating. In a simple cell, the response grows as the intensity of the grating grows, and then falls off as the contrast of the grating reverses. The negative response is weak, because the cell's resting output is low so that it cannot code much inhibition. The response is what one would expect from a linear system with a lower threshold on its response. On the right, the response of a complex cell, which looks like full wave rectification; the cell responds similarly to a grating with positive and reversed contrast.

the positive half of the amplitude signal — and a complex cell as performing full wave rectification — it gives a response to both the positive and negative half of the amplitude signal.

9.2.6 A Model of Early Spatial Vision

We now have a picture of the early stages of primate vision. The retinal image is transformed into a series of retinotopic maps, each of which contains the output of a linear filter which may have spatial or spatio-temporal support. The retinotopic structure means that each map can be thought of as an image which is filtered version of the retinal image. The filters themselves are oriented filters that look rather a lot like various derivatives of a Gaussian, at various orientations. The retinotopic maps are subjected to some form of non-linearity (to get the output of the complex cells).

This model can be refined somewhat, with the aid of psychophysical studies of adaptation. Adaptation is a term that is used fairly flexibly; generally, the response of an observer to a stimulus declines if the stimulus is maintained and stays depressed for some time afterwards. Adaptation can be used to determine components of a signal that are coded differently — or follow different psychophysical channels, in the jargon — if we adopt the model that channels adapt independently. Observer sensitivity to gratings can be measured by the **contrast sensitivity function**, which codes the contrast that a periodic signal must have with a constant background before it is visible (figure **??**).

In experiments by Blakemore and Campbell [], observers were shown spatial frequency gratings until they are adapted to that spatial frequency. It turns out that the observer's contrast sensitivity is decreased for a range of spatial frequencies around the adapting frequency. This suggests that the observer is sensitive to several spatial frequency channels; the contrast sensitivity function can be seen as a superposition of several contrast sensitivity functions, one for each channel.

This is a **multiresolution model**. The current best model of human early vision is that the visual signal is split into several spatial frequency bands (rather as in section **??**); each band is then subjected to a set of oriented linear filters, and the responses of these filters in turn are subjected to a non-linearity (as in figure 9.8). The response of this model can be used quite successfully to predict various forms of pattern sensitivity for simple patterns (it clearly doesn't explain, say, recognition); we will see it again in discussions of texture.

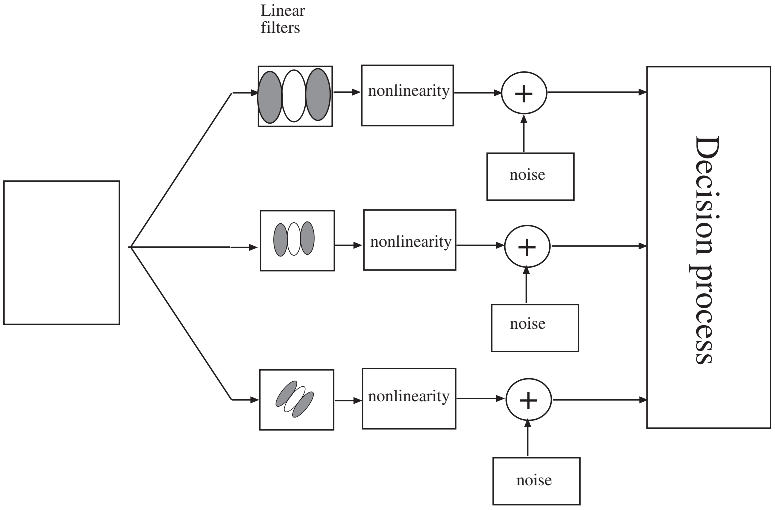


Figure 9.8. An overview of a multiresolution model of human pattern sensitivity. The stimulus is convolved with linear filters at a variety of scales and orientations — we show three scales, and only one orientation per scale; this is *not* a commitment to a number — and then subjected to a nonlinearity. The results have noise added, and are passed to a decision process. The details of the number of scales, the number of orientations, the choice of non-linearity, etc. vary from author to author. This class of model is now quite successful at predicting responses to simple patterns. (after *Figure from Brian Wandell's book, "Foundations of Vision", page222, in the fervent hope that permission will be granted*, who got it from Spillman and Werner)

9.3 Technique: Normalised Correlation and Finding Patterns

We can think of convolution as comparing a filter with a patch of image centered at the point whose response we are looking at. In this view, the image neighbourhood corresponding to the filter kernel is scanned into a vector which is compared with the filter kernel. By itself, this dot-product is a poor way to find features, because the value may be large simply because the image region is bright. By analogy with vectors, we are interested in the cosine of the angle between the filter vector and the image neighbourhood vector; this suggests computing the root sum of squares of the relevant image region (the image elements that would lie under the filter kernel) and dividing the response by that value.

This yields a value that is large and positive when the image region looks like the filter kernel, and small and negative when the image region looks like a contrast-reversed version of the filter kernel. This value could be squared if contrast reversal doesn't matter. This is a cheap and effective method for finding patterns, often called **normalised correlation**.

9.3.1 Controlling the Television by Finding Hands by Normalised Correlation

It would be nice to have systems that could respond to human gestures. You might, for example, wave at the light to get the room illuminated, or point at the airconditioning to get the temperature changed. In typical consumer applications, there are quite strict limits to the amount of computation available, meaning that it is essential that the gesture recognition system be simple. However, such systems are usually quite limited in what they need to do, too.

Typically, a user interface is in some state — perhaps a menu is displayed — and an event occurs — perhaps a button is pressed on a remote control. This event causes the interface to change state — a new menu item is highlighted, say — and the whole process continues. In some states, some events cause the system to perform some action — the channel might change. All this means that a state machine is a natural model for a user interface.

One way for vision to fit into this model is to provide events. This is good, because there are generally very few different kinds of event, and we know what kinds of event the system should care about in any particular state. As a result, the vision system needs only to determine whether either nothing or one of a small number of known kinds of event has occurred. It is quite often possible to build systems that meet these constraints.

Controlling the Television

A relatively small set of events is required to simulate a remote control — one needs events that "look like" button presses (for example, to turn the television on or off), and events that "look like" pointer motion (for example, to increase the volume; it is possible to do this with buttons, too). With these events, the television can be

Missing
Figure

Figure 9.9. Freeman's system controlling his telly

turned on, and an on-screen menu system navigated.

Freeman *et al.* produced an interface where an open hand turns the television on. This can be robust, because all the system needs to do is determine whether there is a hand in view. Furthermore, the user will cooperate by holding their hand up and open. Because the user is expected to be a fairly constant distance from the camera — so the size of the hand is roughly known, and there is no need to search over scales — and in front of the television, the image region that needs to be searched to determine if there is a hand is quite small.

The hand is held up in a fairly standard configuration and orientation to turn the television set on (so we know what it will look like). This means that Freeman can get away with using a normalised correlation score to find the hand. Any points in the correlation image where the score is high enough correspond to hands.

This approach can be used to control volume, etc. as well as turn the television on and off. To do so, we need some notion of where the hand is going — to one side turns the volume up, to the other turns it down — and this can be obtained by comparing the position in the previous frame with that in the current frame. The system displays an iconic representation of its interpretation of hand position, so the user has some feedback as to what the system is doing (figure 9.9).

9.4 Corners and Orientation Representations

Edge detectors notoriously fail at corners, because the assumption that estimates of the partial derivatives in the x and y direction suffice to estimate an oriented gradient becomes unsupportable. At sharp corners or unfortunately oriented corners, these partial derivative estimates will be poor, because their support will cross the corner. There are a variety of specialised corner detectors, which look for image neighbourhoods where the gradient swings sharply. More generally, the statistics of the gradient in an image neighbourhood yields quite a useful description of the

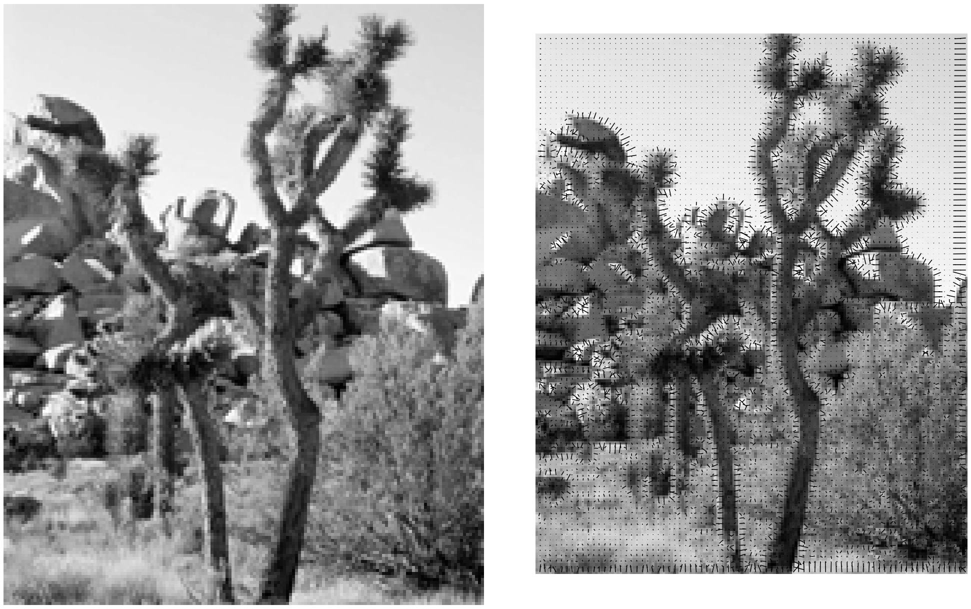


Figure 9.10. An image of a joshua tree, and its orientations shown as vectors superimposed on the image. The center image shows the orientation superimposed on top of the image as small vectors. Notice that around corners and in textured regions, the orientation vector swings sharply.

neighbourhood. There is a rough taxonomy of four qualitative types of image window:

- *constant windows*, where the grey level is approximately constant;
- *edge windows*, where there is a sharp change in image brightness that runs along a single direction within the window;
- *flow windows*, where there are several fine parallel stripes — say hair or fur — within the window;
- and *2D windows*, where there is some form of 2D texture — say spots, or a corner — within the window.

These cases correspond to different kinds of behaviour on the part of the image gradient. In constant windows, the gradient vector is short; in edge windows, there is a small number of long gradient vectors all pointing in a single direction; in flow windows, there are many gradient vectors, pointing in two directions; and in 2D windows, the gradient vector swings.

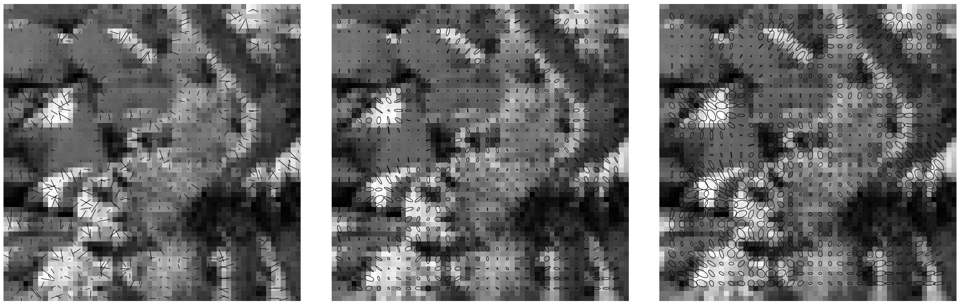


Figure 9.11. The orientation field for a detail of the joshua tree picture. On the left, the orientations shown as vectors and superimposed on the image. Orientations have been censored to remove those where the gradient magnitude is too small. The center shows the ellipses described in the text, for a 3x3 window; left shows the ellipses for a 5x5 window.

These distinctions can be quite easily drawn by looking at variations in orientation within a window. In particular, the matrix

$$\mathcal{H} = \sum_{window} \left\{ (\nabla I)(\nabla I)^T \right\} \approx \sum_{window} \left\{ \begin{array}{cc} (\frac{\partial G_\sigma}{\partial x} ** \mathcal{I})(\frac{\partial G_\sigma}{\partial x} ** \mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} ** \mathcal{I})(\frac{\partial G_\sigma}{\partial y} ** \mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} ** \mathcal{I})(\frac{\partial G_\sigma}{\partial y} ** \mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} ** \mathcal{I})(\frac{\partial G_\sigma}{\partial y} ** \mathcal{I}) \end{array} \right\}$$

gives a good idea of the behaviour of the orientation in a window. In a constant window, both eigenvalues of this matrix will be small, because all terms will be small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients will run in other directions. In a flow window, we expect the same properties of the eigenvalues, except that the large eigenvalue is likely to be larger because many edges contribute. Finally, in a 2D window, both eigenvalues will be large.

The behaviour of this matrix is most easily understood by plotting the ellipses

$$(x, y)^T \mathcal{H}^{-1} (x, y) = \epsilon$$

for some small constant ϵ. These ellipses are superimposed on the image windows. Their major and minor axes will be along the eigenvectors of $\mathcal{H}$, and the extent of the ellipses along their major or minor axes corresponds to the size of the eigenvalues; this means that a large circle will correspond to an edge window and a narrow extended ellipse will indicate an edge window as in figure 9.10. Thus, corners could be marked by marking points where the area of this ellipse is large. The localisation accuracy of this approach is limited by the size of the window and the behaviour of the gradient. More accurate localisation can be obtained, at the price of providing a more detailed model of the corner sought (see, for example, []).

9.5 Advanced Smoothing Strategies and Non-linear Filters

Another way to think of a smoothing filter is as a statistical estimator. In particular, the goal here is to estimate the actual image value at a pixel, in the presence of noisy measurements. This view leads us to a class of filters that are hard to analyse, but can be extremely useful. Typically, these filters are used in circumstances where the stationary additive Gaussian noise model is unusable. We describe a variety of more complex noise models which occur in practice, and then discuss filters that can be used to smooth away these models.

9.5.1 More Noise Models

Salt and pepper noise models cameras with defective sample sites. Pixels are chosen uniformly at random; each of these pixels is set to be either full value or zero value (again, uniformly at random). The result looks as though the image has been sprinkled with salt and pepper, whence the name. There is a basic conceptual difference between stationary additive Gaussian noise and salt and pepper noise; in the first case, we add a random quantity to each pixel, whereas in the second, we use a random mechanism to select pixels, and then operate on the selected pixels.

Random mechanisms to select pixels are usually called **point processes**, and form a significant topic of their own. We will describe some important types of point process. In a **homogenous Poisson point process**, points on the image plane are chosen randomly so that the expected number of points in any subset is proportional to the area of the subset. The constant of proportionality is known as the intensity of the process. An instance of a Poisson point process can be obtained by sampling the number of affected pixels from a Poisson distribution whose mean is the intensity times the image area, and then drawing the coordinates of these pixels uniformly at random.

Because we need to flip some pixels to white and other to black, we need to use a **marked point process**. In this model, we use a point process to select points, then assign to each point a "mark" (for example, whether it is white or black, or some other such thing) at random using an appropriate distribution. For a camera where a non-responsive pixel is as likely as a saturated pixel, the probability that a point carries a black mark should be the same as the probability that a point carries a white mark. If (for example, because of the manufacturing process) there are fewer non-responsive pixels than responsive pixels, then the distribution on the marks can reflect this as well.

A noise process of this form results in fairly evenly distributed noise. A set of bad pixels that consists of widely separated large, tight clumps is quite unlikely. One model that would achieve this takes points chosen by a Poisson process and then marks a clump of pixels around them. The shape of the clump is chosen randomly — this is another form of mark. Another form of noise that is quite common in videotape systems involves whole scan-lines of an image being turned to noise. The line involved can be chosen with a Poisson point process as well (figure 9.13). A

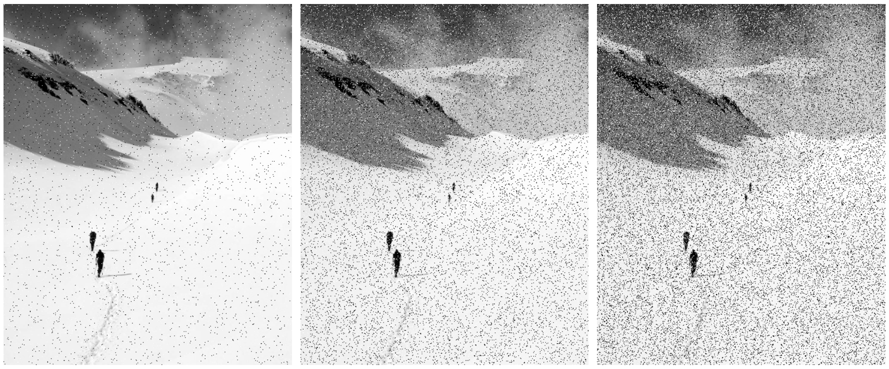


Figure 9.12. Examples of salt-and-pepper noise at different intensities (Poisson process). The snowy background means that, in some areas, the pepper is more visible than the salt.

variety of models of this form are available; a brief exposition appears in chapter *** of [**?**]; more detail on point processes appears in [**?**].

The response of a filter to spatial noise models of this form is usually impossible to compute analytically. Instead, we rely on simulation. The basic idea is to set up a probabilistic noise model, draw a large number of samples from that model, and then apply the filter in question to the samples. One computes appropriate statistics from the result — for example, the mean and variance of the noise response. In principle it is possible to choose filters from families of filters in this way, although we are not aware of anyone doing so in the computer vision literature.

9.5.2 Robust Estimates

Smoothing an image with a symmetric Gaussian kernel replaces a pixel with some weighted average of its neighbours. If an image has been corrupted with stationary additive zero-mean Gaussian noise, then this weighted average gives a reasonable estimate of the original value of the pixel. The expected noise response is zero, and the estimate has better behaviour in terms of spatial frequency than a simple average (as the ringing effects in figure **??** show).

However, if the image noise is not stationary additive Gaussian noise, difficulties arise. For example, consider a noise model where image points are set to the brightest or darkest possible value with a Poisson point process (section 9.14). In particular, consider a region of the image which has a constant dark value and there is a single bright pixel due to noise — smoothing with a Gaussian will leave a smooth, Gaussian-like, bright bump centered on this pixel.

The problem here is that a weighted average can be arbitrarily badly affected by very large noise values. Thus, in our example, we can make the bright bump arbitrarily bright by making the bright pixel arbitrarily bright — perhaps as result of, say, a transient error in reading a memory element. Estimators that do not have

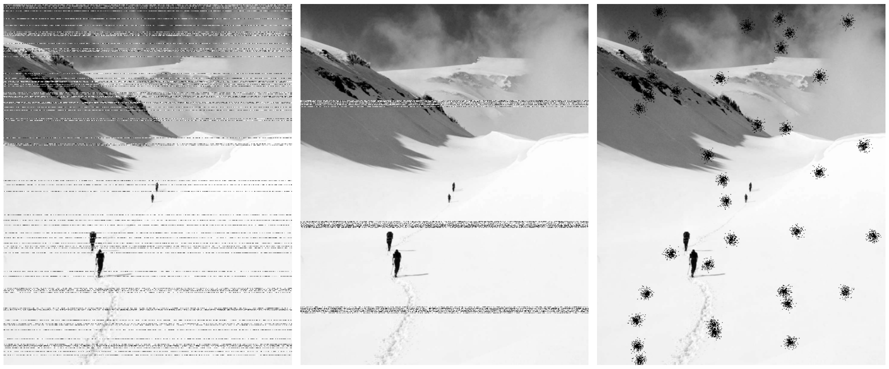


Figure 9.13. A variety of other natural spatial noise models. We show two versions. The first image shows the result of randomly replacing lines with noise, where the probability of replacing the line is uniform; in the second, a block of lines of random length is replaced with noise — this process is a reasonable model of the sort of noise that occurs in fast-forwarding or rewinding VCR's. In the third, a Poisson process chooses "noise points"; in a neighbourhood of each noise point, pixels are randomly marked black with a probability that falls off as the negative exponential of the distance from the noise point. This process simulates damage to the CCD array, or to the lens.

this most undesirable property are often known as **robust estimates**.

The best known robust estimator involves estimating the mean of a set of values using its **median**. For a set with $2k+1$ elements, the median is the $k+1$'th element of the sorted set of values. For a set with $2k$ elements, the median is the average of the k and the $k+1$'th element of the sorted set. It does not matter whether the set is sorted in increasing or decreasing order (exercises!).

9.5.3 Median Filters

A **median filter** is specified by giving some form of neighbourhood shape (which can significantly affect the behaviour of the filter). This neighbourhood is passed over the image as in convolution, but instead of taking a weighted sum of elements within the neighbourhood, we take the median. If we write the neighbourhood centered at i, j as N_{ij}, the filter can be described by:

$$y_{ij} = med(\{x_{uv} | x_{uv} \in N_{ij}\})$$

Applying a median filter to our example of a uniform dark region with a single, arbitrarily bright, pixel will yield a dark region. In this example, up to half of the elements in the neighbourhood could be noise values and the answer would still be correct (exercises!). It is difficult to obtain analytic results about the behaviour of median filters, but a number of general observations apply.

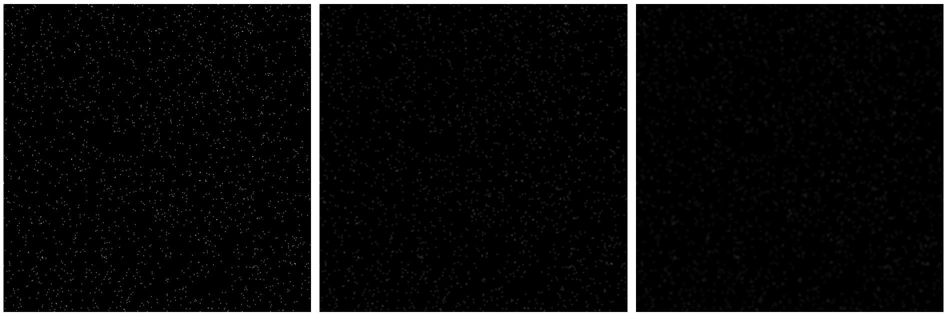


Figure 9.14. On the left, a black background with white noise pixels distributed as a Poisson point process. These pixels are outliers, in the sense that they differ radically from their neighbouring pixels. In the center image, we see the result of estimating pixels as the response of the image to a Gaussian filter with σ one pixel; we are estimating a pixel value as a weighted sum of its neighbours. Because the noise pixels are wildly different from their neighbourhood, they skew this estimate substantially. In the right hand image, we see the result of using a Gaussian filter with σ two pixels; the effect remains, but is smaller, because the effective support of the filter is larger.

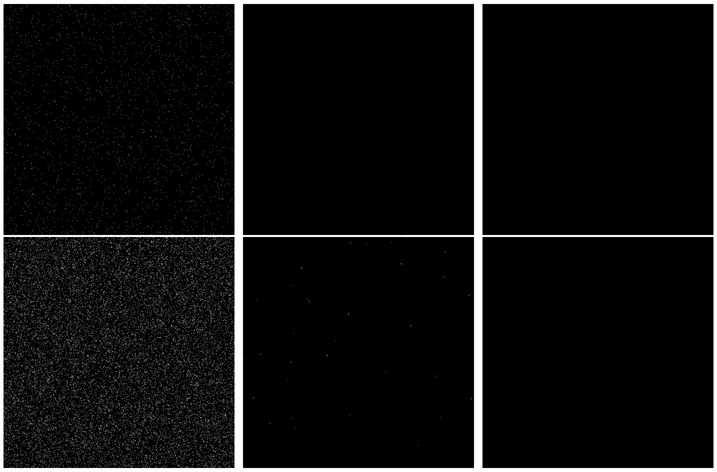


Figure 9.15. The columns on the left show Poisson noise processes of different intensities; on the top row, there are 2000 noise pixels and on the bottom row, 20000. The second column shows the effect of applying a filter that returns the median of a 3x3 neighbourhood to these images, and the third column shows the effect of applying a filter that returns the median of a 7x7 neighbourhood to these images. Notice that, if the noise is intense, then the median filter is unable to suppress it.

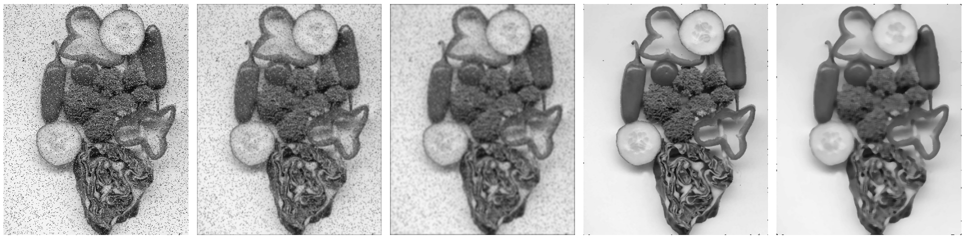


Figure 9.16. On the **left**, an image corrupted with salt-and-pepper noise (points are chosen by a Poisson process, and then with even probability marked either black or white; in this image, about 9% of the pixels are noise pixels). Gaussian smoothing (**center left** shows σ one pixel and **center** shows σ two pixels) works particularly poorly, as the contrast makes the dark regions left behind by averaging in dark pixels very noticeable. A median filter is much more successful (**center right** shows a 3x3 median filter and **right** shows a 7x7 median filter). Notice how the median filter blurs boundaries.

Multi-stage Median Filters

Median filters preserve straight edges, but tend to behave badly at sharp corners (figure 7.1 and exercises). This difficulty is usually dealt with by forming a **multi-stage median filter**; this filter responds with the median of a set of different medians, obtained in different neighbourhoods:

$$\begin{aligned} y_{ij} &= med(z_1, z_2, z_3, z_4) \\ z_1 &= med(\{x_{uv}|x_{uv} \in N^1_{ij}\}) \\ z_2 &= med(\{x_{uv}|x_{uv} \in N^2_{ij}\}) \\ z_3 &= med(\{x_{uv}|x_{uv} \in N^3_{ij}\}) \\ z_4 &= med(\{x_{uv}|x_{uv} \in N^4_{ij}\}) \end{aligned}$$

where N^1 is a vertically extended neighbourhood, N^2 is a horizontally extended neighbourhood, and N^3 and N^4 are diagonal neighbourhoods. Exercise **??** asks for an intuitive argument as to why this filter is inclined to preserve corners; the effect is illustrated in figure 9.17.

Trimmed and Hybrid Median Filters

While median filters tend to be better than linear filters at rejecting very large noise values — so called **outliers** — they tend to be poorer than linear filters at handling noise that does not have outliers. In jargon, noise that can produce occasional large values is often called **long-tailed noise**, because the probability density for the noise values has "long tails" —there is significant weight in the density far from the mean; similarly, noise that does not have this property is often called **short-tailed noise**. In a neighbourhood, long-tailed noise will produce a small number of very large values, which tend not to affect the median much; however, short-tailed

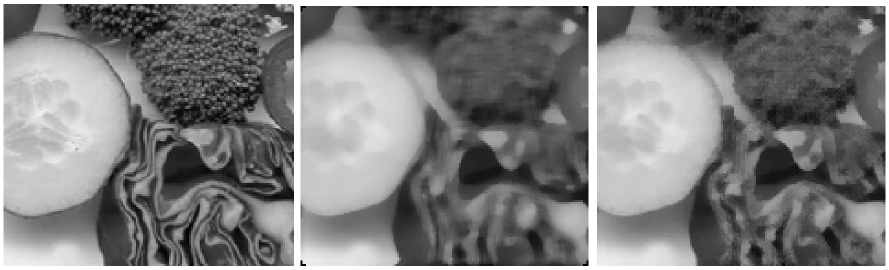


Figure 9.17. On the left, a detail from the figure of vegetables; in the center the result of applying a median filter with a 7x7 neighbourhood of support. Notice that the texture of the broccoli florets is almost completely smoothed away, and that corners around the red cabbage have been obscured. These effects could be useful in some contexts, but reduce the usefulness of the filter in suppressing long-tailed noise because they represent a reduction in image detail, too. On the right, the result of applying a multistage median filter, using 7 pixel domains that are horizontal, vertical, and along the two diagonals. Significantly less detail has been lost.

noise will produce values that are similar to the original pixel value and which will affect the median more. This difficulty can be handled either by using an **α-trimmed linear filter** — where $\alpha/2$ percent of the largest and smallest values in a neighbourhood are removed from consideration and the rest are subjected to a linear filter — or by using a **hybrid median filter** — where the output is the median of a set of linear filters over a neighbourhood.

Median filters can be extremely slow. One strategy is to pretend that a median filter is separable, and apply separate x and y median filters.

9.5.4 Mathematical morphology: erosion and dilation

A variety of useful operators can be obtained from considering set-theoretic operations on binary images. It often occurs that a binarised images has individual pixels or small groups of pixels that are isolated from the main body of the image. Commonly, one would like to remove very small groups of pixels and join up groups that are close together. Small groups can be removed by noticing that a block of pixels would not fit inside a small group; large groups can be joined up by "thickening" their boundaries. In figure 9.18, we illustrate removing groups of dark pixels by removing pixels that are not at the center of a 3x3 block of dark pixels (i.e. pixels where some neighbour is light). Similarly, gaps can be jumped by attaching a 3x3 neighbourhood to each pixel.

These (quite useful) tricks can be generalised. For example, there is no need to insist on a 3x3 neighbourhood — any pattern will do. The generalisation is most easily phrased in terms of sets. Assume, for the moment, we have two binary images $\mathcal{I}$ and $\mathcal{S}$ (i.e. pixel values in each can be only either 0 or 1). We can regard each image as a representation of a set of elements which belong to a finite grid. Pixels

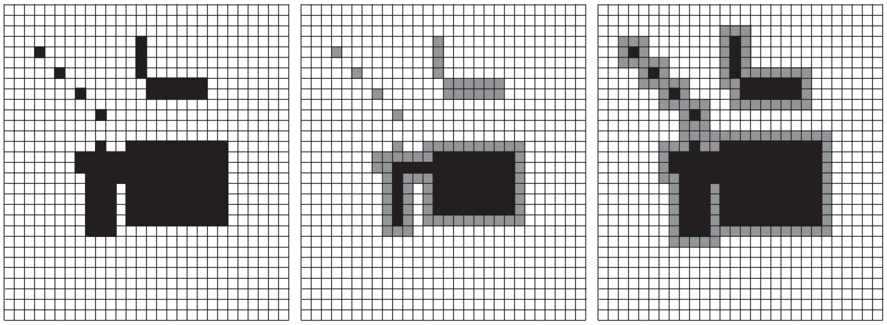


Figure 9.18. On the left, a binary image; a natural strategy for removing small groups of dark pixels is to lighten all pixels that do *not* lie at the center of a 3x3 dark neighbourhood. This process is known as erosion. In the center, the relevant pixels have been greyed. Similarly, we could fill in small gaps by marking all pixels such that a 3x3 neighbourhood around the pixel contacts a dark pixel, a process known as dilation. The relevant pixels have been greyed on the right.

that have the value 1 are elements of the set and pixels that have the value 0 are not. Now write $\mathcal{S}_p$ for the image obtained by shifting the center of $\mathcal{S}$ to the pixel p. We can define a new set

$$\mathcal{I} \oplus \mathcal{S} = \{p : \mathcal{S}_p \cap \mathcal{I} \neq \emptyset\}$$

This is called the **dilation** of the set $\mathcal{I}$ by $\mathcal{S}$. Similarly, we can define

$$\mathcal{I} \ominus \mathcal{S} = \{p : \mathcal{S}_p \subset \mathcal{I}\}$$

which is called the **erosion** of the set $\mathcal{I}$ by $\mathcal{S}$. In these operations, $\mathcal{S}$ is usually called the **structuring element**. The properties of these operators have been widely studied []; some are explored in the exercises. Their main application in practice appears in cleaning up data sets. Typically, a predicate is available that marks "interesting" pixels — which might be skin-coloured, or red, or textured, etc. Usually, small groups of pixels pass this criterion as well as the real regions of interest. A few passes of erosion by a 3x3 neighbourhood, followed by a few passes of dilation by a 3x3 neighbourhood, will remove small groups, fill gaps, and leave an estimate of the real region of interest that is often significantly improved. Occasionally, applications arise where erosion or dilation by structuring elements different from kxk neighbourhoods is appropriate [].

9.5.5 Anisotropic Scaling

One important difficulty with scale space models is that the symmetric Gaussian smoothing process tends to blur out edges rather two aggressively for comfort. For example, if we have two trees near one another on a skyline, the large scale blobs corresponding to each tree may start merging before all the small scale blobs have finished. This suggests that we should smooth differently at edge points than at other points. For example, we might make an estimate of the magnitude and orientation of the gradient: for large gradients, we would then use an oriented smoothing operator that smoothed aggressively perpendicular to the gradient and very little along the gradient; for small gradients, we might use a symmetric smoothing operator. This idea used to be known as **edge preserving smoothing**.

In the modern, more formal version (details in in []), we notice the scale space representation family is a solution to the **diffusion equation**

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} \\ &= \nabla^2 \Phi \\ \Phi(x, y, 0) &= \mathcal{I}(x, y)\end{aligned}$$

If this equation is modified to have the form

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \nabla \cdot (c(x, y, \sigma) \nabla \Phi) \\ &= c(x, y, \sigma) \nabla^2 \Phi + (\nabla c(x, y, \sigma)) \cdot (\nabla \Phi) \\ \Phi(x, y, 0) &= \mathcal{I}(x, y)\end{aligned}$$

then if $c(x, y, \sigma) = 1$, we have the diffusion equation we started with, and if $c(x, y, \sigma) = 0$ there is no smoothing. We will assume that c does not depend on σ. If we knew where the edges were in the image, we could construct a mask that consisted of regions where $c(x, y) = 1$, isolated by patches along the edges where $c(x, y) = 0$; in this case, a solution would smooth *inside* each separate region, but not over the edge. While we do not know where the edges are — the exercise would be empty if we did — we can obtain reasonable choices of $c(x, y)$ from the magnitude of the image gradient. If the gradient is large, then c should be small, and vice-versa.

9.6 Commentary

Assignments

Exercises

1. For a set with $2k+1$ elements, the median is the $k+1$'th element of the sorted set of values. For a set with $2k$ elements, the median is the average of the k and the $k+1$'th element of the sorted set. Show that it does not matter whether the set is sorted in increasing or decreasing order.

2. Assume that we wish to remove salt-and-pepper noise from a uniform background. Show that up to half of the elements in the neighbourhood could be noise values and a median filter would still give the same (correct) answer.

Programming Assignments

1. Build a normalised correlation matcher. The hand finding application is a nice one, but another may occur to you.
    - How reliable is it?
    - How many different patterns can you tell apart in practice?
    - How sensitive is it to illumination variations? shadows? occlusion?
2. Median filters can smooth corners unacceptably.
    - Demonstrate this effect by applying a median filter to a variety of images; what are the qualitative effects at corners and in textured regions?
    - Explain these effects.
    - Show that the multi-stage median filter results in less heavily smoothed corners in practice.
    - Explain this effect.
    - On occasion, it is attractive to suppress texture; median filters can do this rather well. Read [**?**].

Chapter 10

TEXTURE

Texture is a phenomenon that is widespread, easy to recognise and hard to define. Typically, whether an effect is referred to as texture or not depends on the scale at which it is viewed. A leaf that occupies most of an image is an object, but the foliage of a tree is a texture. Texture arises from a number of different sources. Firstly, views of large numbers of small objects are often best thought of as textures. Examples include grass, foliage, brush, pebbles and hair. Secondly, many surfaces are marked with orderly patterns that look like large numbers of small objects. Examples include: the spots of animals like leopards or cheetahs; the stripes of animals like tigers or zebras; the patterns on bark, wood and skin.

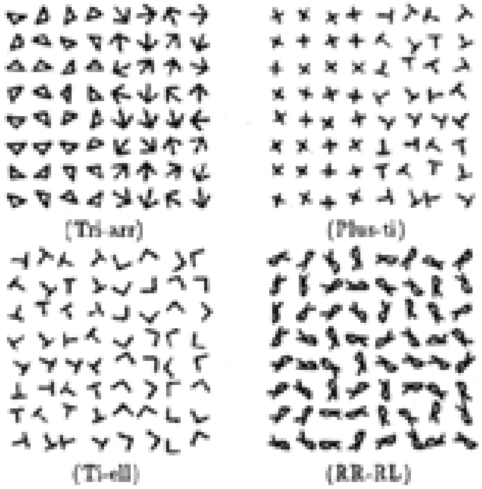


Figure 10.1. A set of texture examples, used in experiments with human subjects to tell how easily various types of textures can be discriminated. Note that these textures are made of quite stylised subelements, repeated in a meaningful way. *figure from the Malik and Perona, A Computational Model of Texture Segmentation, p.331, in the fervent hope, etc.*

There are three standard problems to do with texture:

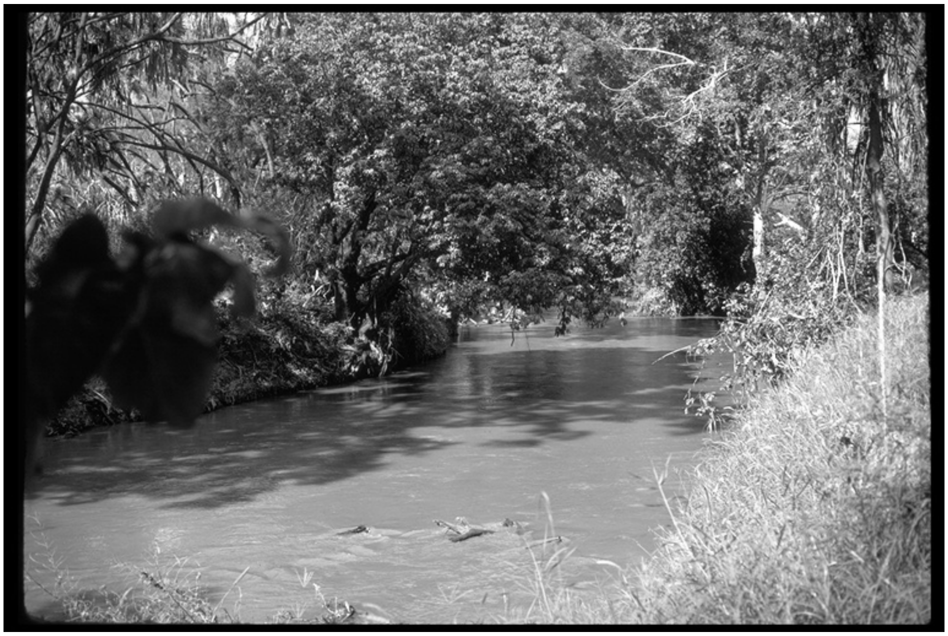


Figure 10.2. A typical textured image. For materials such as brush, grass, foliage and water, our perception of what the material is is quite intimately related to the texture. These textures are also made of quite stylised subelements, arranged in a pattern. *figure from the Malik and Perona, A Computational Model of Texture Segmentation, p.331, in the fervent hope, etc. figure from the Calphotos collection, number. 0057, in the fervent hope, etc.*

- **Texture segmentation** is the problem of breaking an image into components within which the texture is constant. Texture segmentation involves both representing a texture, and determining the basis on which segment boundaries are to be determined. In this chapter, we deal only with the question of how textures should be *represented* (section 10.1); chapter **??** shows how to segment textured images using this representation.

- **Texture synthesis** seeks to construct large regions of texture from small example images. We do this by using the example images to build probability models of the texture, and then drawing on the probability model to obtain textured images. There are a variety of methods for building a probability model; three successful current methods are described in section 10.3.

- **Shape from texture** involves recovering surface orientation or surface shape from image texture. We do this by assuming that texture "looks the same" at different points on a surface; this means that the deformation of the texture from point to point is a cue to the shape of the surface. In section 10.4 and section **??**, we describe the main lines of reasoning in this (rather technical) area.

10.1 Representing Texture

Image textures generally consist of organised patterns of quite regular subelements (sometimes called **textons**). For example, one texture in figure 10.1 consists of triangles. Similarly, another texture in that figure consists of arrows. One natural way to try and represent texture is to find the textons, and then describe the way in which they are laid out.

The difficulty with this approach is that there is no known canonical set of textons, meaning that it isn't clear what one should look for. Instead of looking for patterns at the level of arrowheads and triangles, we could look for even simpler pattern elements — dots and bars, say — and then reason about their spatial layout. The advantage of this approach is that it is easy to look for simple pattern elements by filtering an image.

10.1.1 Extracting Image Structure with Filter Banks

In section 9.1, we saw that convolving an image with a linear filter yields a representation of the image on a different basis. The advantage of transforming an image to the new basis given by convolving it with a filter, is that the process makes the local structure of the image clear. This is because there is a strong response when the image pattern in a neighbourhood looks similar to the filter kernel, and a weak response when it doesn't.

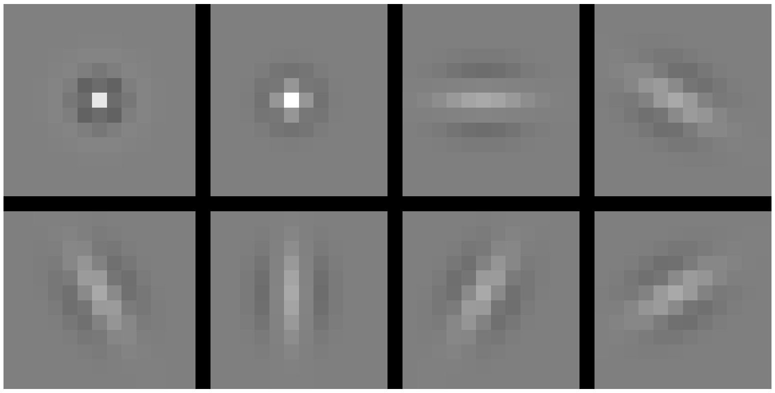


Figure 10.3. A set of eight filters used for expanding images into a series of responses. These filters are shown at a fixed scale, with zero represented by a mid-grey level, lighter values being positive and darker values being negative. They represent two distinct spots, and six bars; the set of filters is that used by [**?**].

This suggests representing image textures in terms of the response of a collection of filters. The collection of different filters would consist of a series of patterns — spots and bars are usual — at a collection of scales (to identify bigger or smaller spots or bars, say). The value at a point in a derived image represents the local "spottiness" ("barriness", etc.) at a particular scale at the corresponding point in the image. While this representation is now heavily redundant, it exposes structure ("spottiness", "barriness", etc., in a way that has proven helpful. The process of convolving an image with a range of filters is referred to as **analysis**.

Generally, spot filters are useful because they respond strongly to small regions that differ from their neighbours (for example, on either side of an edge, or at a spot). The other attraction is that they detect non-oriented structure. Bar filters, on the other hand, are oriented, and tend to respond to oriented structure (this property is sometimes, rather loosely, described as **analysing orientation** or **representing orientation**).

Spots and Bars by Weighted Sums of Gaussians

But what filters should we use? There is no canonical answer. A variety of answers have been tried. By analogy with the human visual cortex, it is usual to use at least one spot filter and a collection of oriented bar filters at different orientations, scales and **phases**. The phase of the bar refers to the phase of a cross-section perpendicular to the bar, thought of as a sinusoid (i.e. if the cross section passes through zero at the origin, then the phase is 0^o.

One way to obtain these filters is to form a weighted difference of Gaussian filters at different scales; this technique was used for the filters of figure 10.3. The filters for this example consist of

- **A spot**, given by a weighted sum of three concentric, symmetric Gaussians, with weights 1, -2 and 1, and corresponding sigmas 0.62, 1 and 1.6.

- **Another spot**, given by a weighted sum of two concentric, symmetric Gaussians, with weights 1 and -1, and corresponding sigmas 0.71 and 1.14.

- **A series of oriented bars**, consisting of a weighted sum of three oriented Gaussians, which are offset with respect to one another. There are six versions of these bars; each is a rotated version of a horizontal bar. The Gaussians in the horizontal bar have weights -1, 2 and -1. They have different sigma's in the x and in the y directions; the σ_x values are all 2, and the σ_y values are all 1. The centers are offset along the y axis, lying at $(0, 1)$, $(0, 0)$ and $(0, -1)$.

You should understand that the details of the choice of filter are almost certainly immaterial. There is a body of experience that suggests that there should be a series of spots and bars at various scales and orientations — which is what this collection provides — but very little reason to believe that optimising the choice of filters produces any major advantage.

Figures 10.4 and 10.5 illustrate the absolute value of the responses of this bank of filters to an input image of a butterfly. Notice that, while the bar filters are not

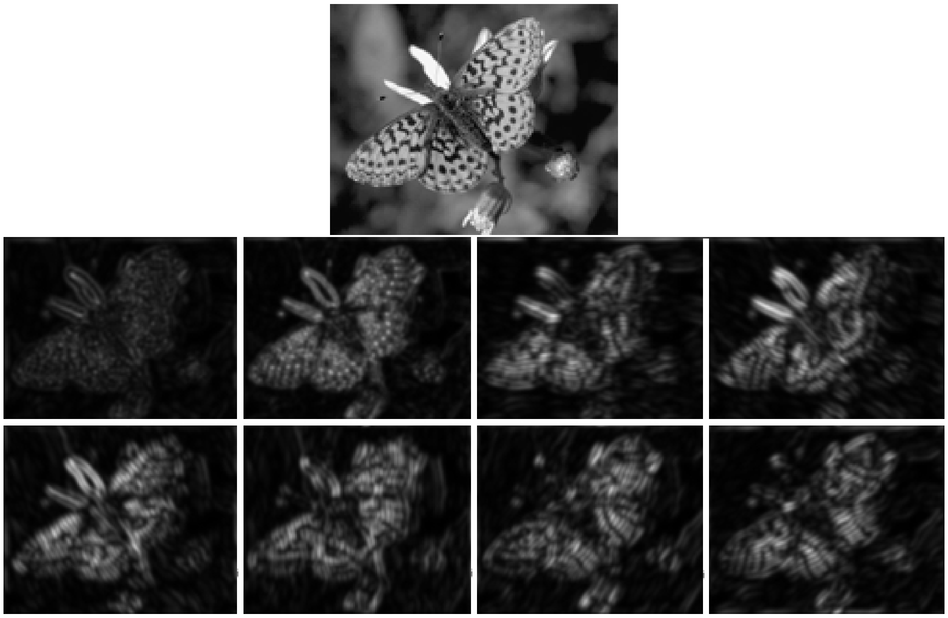


Figure 10.4. At the top, an image of a butterfly at a fine scale, and below, the result of applying each of the filters of figure 10.3 to that image. The results are shown as absolute values of the output, lighter pixels representing stronger responses, and the images are laid out corresponding to the filter position in the top row.

completely reliable bar detectors (because a bar filter at a particular orientation responds to bars of a variety of sizes and orientations), the filter outputs give a reasonable representation of the image data. Generally, bar filters respond strongly to oriented bars and weakly to other patterns, and the spot filter responds to isolated spots.

Spots and Bars by Gabor Filters

Another way to build spot and bar filters is to use **Gabor filters**. The kernels look like Fourier basis elements that are multiplied by Gaussians, meaning that a Gabor filter responds strongly at points in an image where there are components that *locally* have a particular spatial frequency and orientation. Gabor filters come in pairs, often referred to as **quadrature pairs**; one of the pair recovers symmetric components in a particular direction, and the other recovers antisymmetric components. The mathematical form of the symmetric kernel is

$$G_{\mathrm{S}ymmetric}(x,y) = \cos\left(k_x x + k_y y\right) \exp - \left\{ \frac{x^2+y^2}{2\sigma^2} \right\}$$

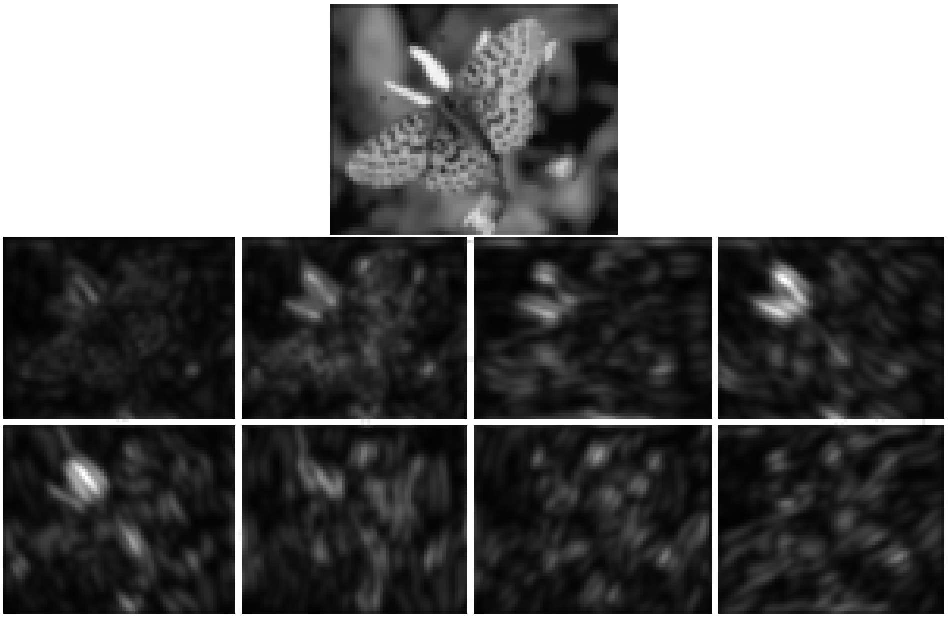


Figure 10.5. The input image of a butterfly and responses of the filters of figure 10.3 at a coarser scale than that of figure 10.4. Notice that the oriented bars respond to the bars on the wings, the antennae, and the edges of the wings; the fact that one bar has responded does not mean that another will not, but the size of the response is a cue to the orientation of the bar in the image.

and the antisymmetric kernel has the form

$$G_{antisymmetric}(x, y) = \sin(k_0 x + k_1 y) \exp - \left\{ \frac{x^2 + y^2}{2\sigma^2} \right\}$$

The filters are illustrated in figures 10.6 and 10.7; (k_x, k_y) give the spatial frequency to which the filter responds most strongly, and σ is referred to as the **scale** of the filter. In principle, by applying a very large number of Gabor filters at different scales, orientations and spatial frequencies, one can analyse an image into a detailed local description.

Gabor filter kernels look rather a lot like smoothed derivative kernels, for different orders of derivative. For example, if the spatial frequency of the Gabor filter is low compared to the scale and the phase is zero, we get a kernel that looks a lot like a derivative of Gaussian filter (top left of figure 10.6); if the phase is $\pi/2$, then the kernel looks a lot like a second derivative of Gaussian filter (bottom left of figure 10.6). Another way to think of Gabor filter kernels is as assemblies of bars — as the spatial frequency goes up compared to the scale, the filter looks for patches of parallel stripes rather than individual bars.

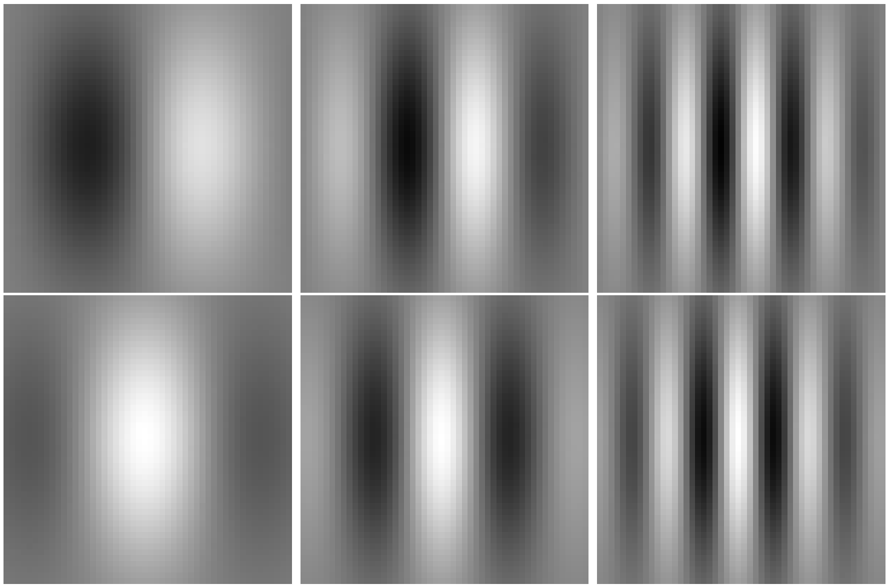


Figure 10.6. Gabor filter kernels are the product of a symmetric Gaussian with an oriented sinusoid; the form of the kernels is given in the text. The images show Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The symmetric and antisymmetric components have a phase difference of $\pi/2$ radians, because a cross-section perpendicular to the bar (horizontally, in this case) gives sinusoids that have this phase difference. The scale of these filters is constant, and they are shown for three different spatial frequencies. Notice how these filters look rather like derivative of Gaussian filters — as the spatial frequency goes up, so does the derivative in the derivative of Gaussian model. It can be helpful to think of these filters as seeking groups of bars. Figure 10.7 shows Gabor filters at a finer scale.

How many Filters and at what Orientation?

It is not known just how many filters are required for useful texture algorithms. Perona lists the number of scales and orientation used in a variety of systems; numbers run from four to eleven scales and from two to eighteen orientations [**?**]. The number of orientations varies from application to application and does not seem to matter much, as long as there are at least about six orientations. Typically, the "spot" filters are Gaussians and the "bar" filters are obtained by differentiating oriented Gaussians.

Similarly, there does not seem to be much benefit in using more complicated sets of filters than the basic spot and bar combination. There is a tension here: using more filters leads to a more detailed (and more redundant representation of

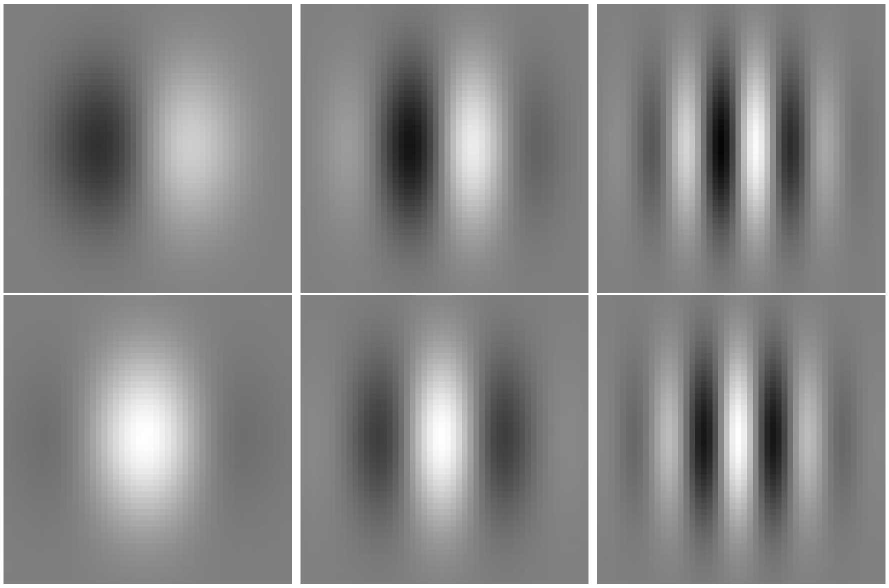


Figure 10.7. The images shows Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The scale of these filters is constant, and they are shown for three different spatial frequencies. These filters are shown at a finer scale than those of figure 10.6.

the image); but we must also convolve the image with all these filters, which can be expensive. Section **??** illustrates a variety of the tricks that are used to reduce the computational expense.

10.2 Analysis (and Synthesis) Using Oriented Pyramids

Analysing images using filter banks presents a computational problem — we have to convolve an image with a large number of filters at a range of scales. The computational demands can be simplified by handling scale and orientation systematically. The Gaussian pyramid (section **??**) is an example of image analysis by a bank of filters — in this case, smoothing filters. The Gaussian pyramid handles scale systematically by subsampling the image once it has been smoothed. This means that generating the next coarsest scale is easier, because we don't process redundant information.

In fact, the Gaussian pyramid is a highly redundant representation because each layer is a low pass filtered version of the previous layer — this means that we are representing the lowest spatial frequencies many times. A layer of the Gaussian pyramid is a prediction of the appearance of the next finer scale layer — this prediction isn't exact, but it means that it is unnecessary to store all of the next finer scale layer. We need keep only a record of the errors in the prediction. This is the motivating idea behind the **Laplacian pyramid**.

The Laplacian pyramid will yield a representation of various different scales that has fairly low redundancy, but it doesn't immediately deal with orientation; in section 10.2.2, we will sketch a method that obtains a representation of orientation as well.

10.2.1 The Laplacian Pyramid

The Laplacian pyramid makes use of the fact that a coarse layer of the Gaussian pyramid predicts the appearance of the next finer layer. If we have an upsampling operator that can produce a version of a coarse layer of the same size as the next finer layer, then we need only store the difference between this prediction and the layer itself.

Clearly, we cannot create image information, but we can expand a coarse scale image by replicating pixels. This involves an upsampling operator $S^{\uparrow}$ which takes an image at level $n+1$ to an image at level n. In particular, $S^{\uparrow}(\mathcal{I})$ takes an image, and produces an image twice the size in each dimension. The four elements of the output image at $(2j-1, 2k-1)$; $(2j, 2k-1)$; $(2j-1, 2k)$; and $(2j, 2k)$ all have the same value as the j, k'th element of $\mathcal{I}$.

Analysis — Building a Laplacian Pyramid from an Image

The coarsest scale layer of a Laplacian pyramid is the same as the coarsest scale layer of a Gaussian pyramid. Each of the finer scale layers of a Laplacian pyramid is a difference between a layer of the Gaussian pyramid and a prediction obtained by upsampling the next coarsest layer of the Gaussian pyramid. This means that:

$$P_{\text{Laplacian}}(\mathcal{I})_m = P_{\text{Gaussian}}(\mathcal{I})_m$$

(where m is the coarsest level) and

$$\begin{aligned} P_{\text{Laplacian}}(\mathcal{I})_k &= P_{\text{Gaussian}}(\mathcal{I})_k - S^{\uparrow}(P_{\text{Gaussian}}(\mathcal{I})_{k+1}) & (10.2.1) \\ &= (Id - S^{\uparrow}S^{\downarrow}G_{\sigma})P_{\text{Gaussian}}(\mathcal{I})_k & (10.2.2) \end{aligned}$$

All this yields algorithm 1. While the name "Laplacian" is somewhat misleading — there are no differential operators here — it is not outrageous, because each layer is approximately the result of a difference of Gaussian filter.

Each layer of the Laplacian pyramid can be thought of as the response of a band-pass filter (that is, the components of the image that lie within a particular range of spatial frequencies. This is because we are taking the image at a particular resolution, and subtracting the components that can be predicted by a coarser resolution version — which corresponds to the low spatial frequency components

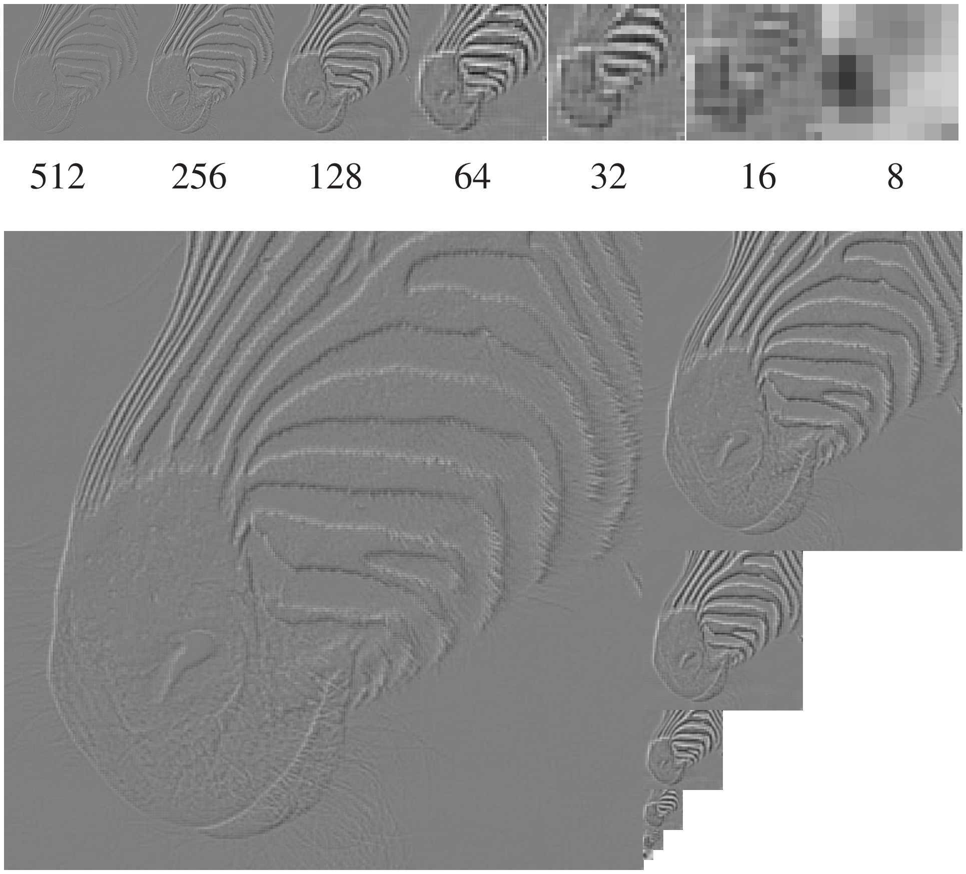


Figure 10.8. A Laplacian pyramid of images, running from 512x512 to 8x8. A zero response is coded with a mid-grey; positive values are lighter and negative values are darker. Notice that the stripes give stronger responses at particular scales, because each layer corresponds (roughly) to the output of a band-pass filter.

of the image. This means in turn that we expect that an image of a set of stripes at a particular spatial frequency would lead to strong responses at one level of the pyramid and weak responses at other levels (figure 10.8).

Because different levels of the pyramid represent different spatial frequencies, the Laplacian pyramid can be used as a reasonably effective image compression scheme. Laplacian pyramids are also used for image blending (figure **??**).

Synthesis — Recovering an Image from its Laplacian Pyramid

Laplacian pyramids have one important feature. It is easy to recover an image from its Laplacian pyramid. We do this by recovering the Gaussian pyramid from

```
Form a Gaussian pyramid

Set the coarsest layer of the Laplacian pyramid to be
the coarsest layer of the Gaussian pyramid

For each layer, going from next to coarsest to finest

  Obtain this layer of the Laplacian pyramid by
  upsampling the next coarser layer, and subtracting
  it from this layer of the Gaussian pyramid

end
```

Algorithm 10.1: *Building a Laplacian pyramid from an image*

the Laplacian pyramid, and then taking the finest scale of the Gaussian pyramid (which is the image) . It is easy to get to the Gaussian pyramid from the Laplacian. Firstly, the coarsest scale of the Gaussian pyramid is the same as the coarsest scale of the Laplacian. The next-to-coarsest scale of the Gaussian pyramid is obtained by taking the coarsest scale, upsampling it, and adding the next-to-coarsest scale of the Laplacian pyramid (and so on up the scales). This process is known as **synthesis** (algorithm 2).

```
Set the working image to be the coarsest layer

For each layer, going from next to coarsest to finest

  upsample the working image and add the current layer
  to the result

  set the working image to be the result of this operation

end
The working image now contains the original image
```

Algorithm 10.2: *Synthesis: obtaining an image from a Laplacian pyramid*

10.2.2 Oriented Pyramids

A Laplacian pyramid does not contain enough information to reason about image texture, because there is no explicit representation of the orientation of the stripes. A natural strategy for dealing with this is to take each layer and decompose it further, to obtain a set of components each of which represents a energy at a distinct orientation. Each component can be thought of as the response of an oriented filter at a particular scale and orientation. The result is a detailed analysis of the image, known as an **oriented pyramid** (figure 10.9).

A comprehensive discussion of the design of oriented pyramids would take us out of our way. However, some insight can be obtained by thinking about pyramids in the Fourier domain. Each layer of the Laplacian pyramid represents the difference between an image convolved with a Gaussian and the same image convolved with a finer scale Gaussian. Convolution in the image domain is equivalent to multiplication in the Fourier domain; the Fourier transform of a Gaussian is a Gaussian; and Fourier transformation is linear. This means that the Fourier transform of each layer of the Laplacian pyramid is obtained by taking the Fourier transform of the image and multiplying it by a difference of Gaussians. This difference of Gaussians will be large within an annulus in Fourier transform space, and small outside this annulus (the left half of figure 10.10). While an ideal bandpass filter would have a unit value within the annulus and a zero value outside, such a filter would have infinite spatial support — making it difficult to work with — and the difference of Gaussians appears to be a satisfactory practical choice. Now if we wish to select orientations as well, we need to modify the Fourier transform of the filter kernel so that it is large within a wedge of the annulus, and small outside (the right half of figure 10.10).

There is a second design constraint for our analysis filters: synthesis should be easy. If we think of the oriented pyramid as a decomposition of the Laplacian pyramid (figure 10.11), then synthesis involves reconstructing each layer of the Laplacian pyramid, and then synthesizing the image from the Laplacian pyramid. The ideal strategy is to have a set of filters that have oriented responses *and* where synthesis is easy. It is possible to produce a set of filters such that reconstructing a layer from its components involves filtering the image a second time with the same filter (as figure 10.12 suggests). An efficient implementation of these pyramids is available at `http://www.cis.upenn.edu/ eero/steerpyr.html`. The design process is described in detail in [].

10.3 Application: Synthesizing Textures for Rendering

Objects rendered using computer graphics systems look more realistic if real textures are rendered on their faces. There are a variety of techniques for texture mapping; the basic idea is that when an object is rendered, the reflectance value used to shade a pixel is obtained by reference to a **texture map**. Some system of coordinates is adopted on the surface of the object to associate the elements of the texture map

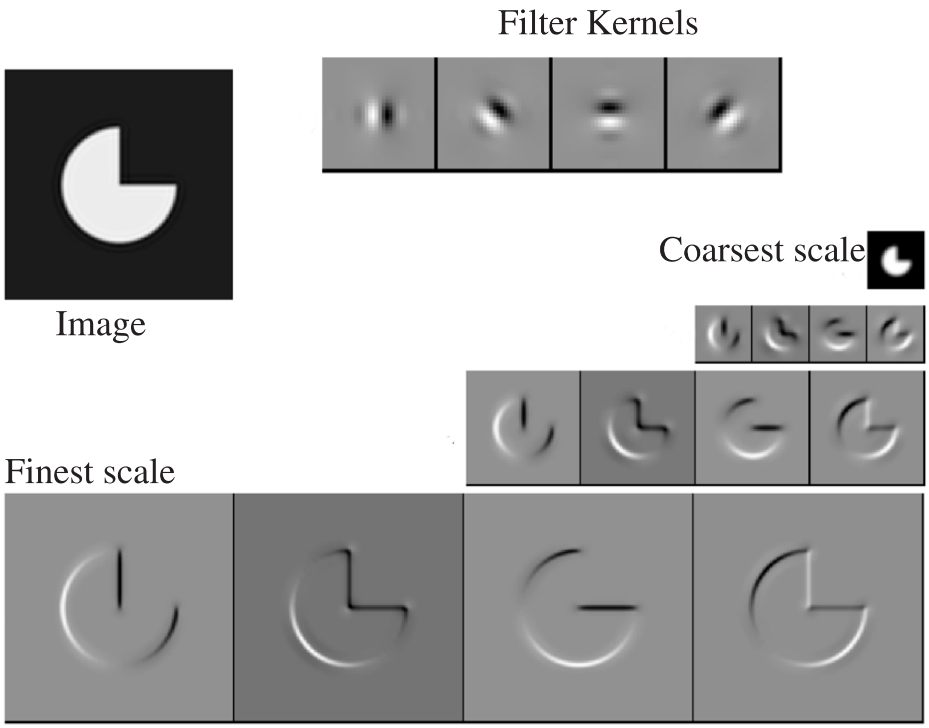


Figure 10.9. An oriented pyramid, formed from the image at the top, with four orientations per layer. This is obtained by firstly decomposing an image into subbands which represent bands of spatial frequency (as with the Laplacian pyramid), and then applying oriented filters to these subbands to decompose them into a set of distinct images, each of which represents the amount of energy at a particular scale and orientation in the image. Notice how the orientation layers have strong responses to the edges in particular directions, and weak responses at other directions. Code for constructing oriented pyramids, written and distributed by Eero Simoncelli, can be found at `http://www.cis.upenn.edu/ eero/steerpyr.html`. *Data obtained from "Shiftable MultiScale Transforms", Simoncelli et al., page 599, in fervent hope that permission will be granted.*

with points on the surface. Different choices of coordinate system yield renderings that look quite different, and it is not always easy to ensure that the texture lies on a surface in a natural way (for example, consider painting stripes on a zebra — where should the stripes go to yield a natural pattern?). Despite this issue, texture mapping seems to be an important trick for making rendered scenes look more realistic.

Texture mapping demands textures, and texture mapping a large object may

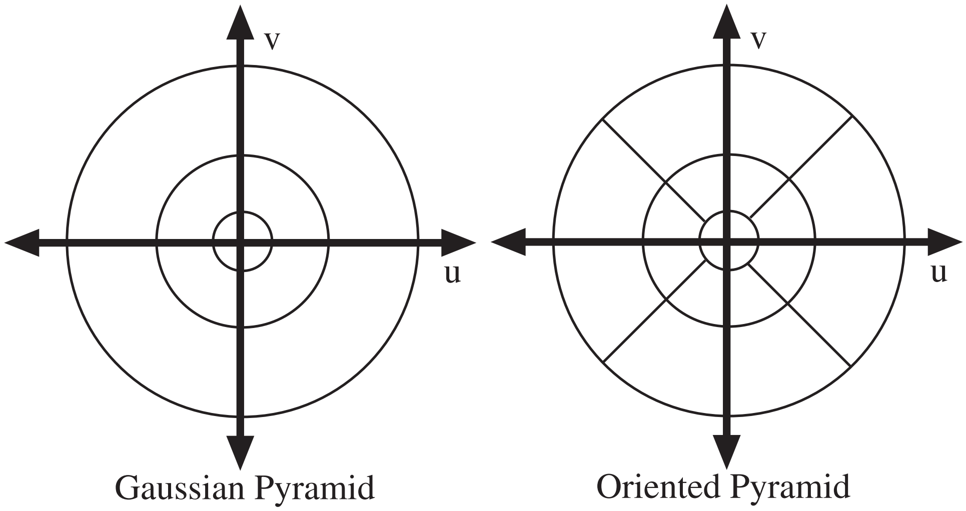


Figure 10.10. Each layer of the Laplacian pyramid consists the elements of a smoothed and resampled image that are not represented by the next smoother layer. Assuming that a Gaussian is a sufficiently good smoothing filter, each layer can be thought of as representing the image components within a range of spatial frequencies — this means that the Fourier transform of each layer is an annulus of values from the Fourier transform space (u, v) space (recall that the magnitude of (u, v) gives the spatial frequency). The sum of these annuluses is the Fourier transform of the image, so that each layer cuts an annulus out of the image's Fourier transform. An oriented pyramid cuts each annulus into a set of wedges. If (u, v) space is represented in polar coordinates, each wedge corresponds to an interval of radius values and an interval of angle values (recall that $\arctan(u/v)$ gives the orientation of the Fourier basis element).

require a substantial texture map. This is particularly true if the object is close to the view, meaning that the texture on the surface is seen at a high resolution, so that problems with the resolution of the texture map will become obvious. Tiling texture images can work poorly, because it can be difficult to obtain images that tile well — the borders have to line up, and even if they did, the resulting periodic structure can be annoying. It is possible to buy image textures from a variety of sources, but an ideal would be to have a program that can generate large texture images from a small example. Quite sophisticated programs of this form can be built, and they illustrate the usefulness of representing textures by filter outputs.

10.3.1 Homogeneity

The general strategy for texture synthesis is to think of a texture as a sample from some probability distribution and then to try and obtain other samples from that same distribution. To make this approach practical, we need to obtain a probability

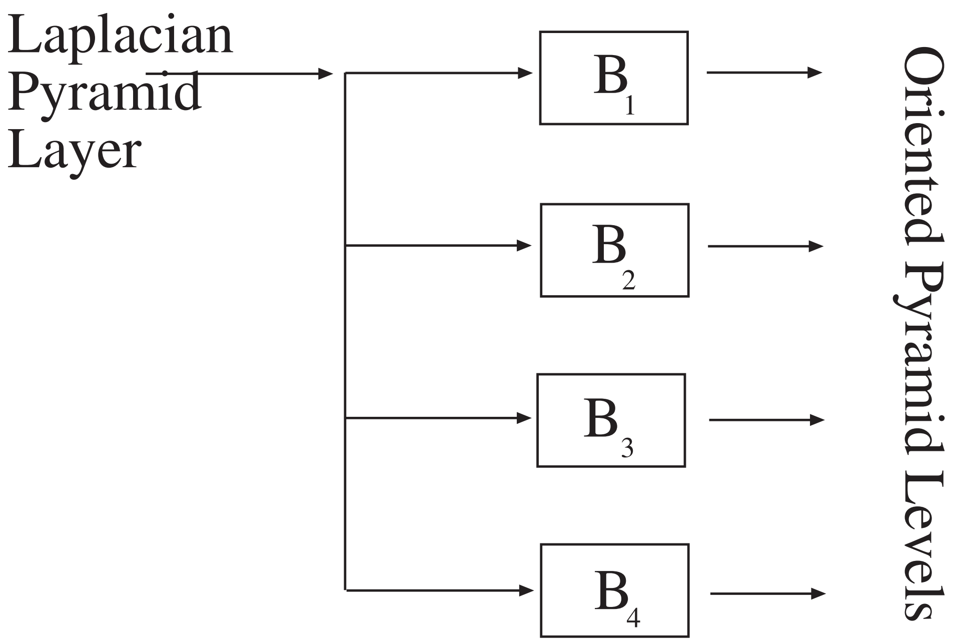


Figure 10.11. The oriented pyramid is obtained by taking layers of the Laplacian pyramid, and then applying oriented filters (represented in this schematic drawing by boxes). Each layer of the Laplacian pyramid represents a range of spatial frequencies; the oriented filters decompose this range of spatial frequencies into a set of orientations.

model. The first thing to do is assume that the texture is **homogenous**. This means that local windows of the texture "look the same", from wherever in the texture they were drawn. More formally, the probability distribution on values of a pixel is determined by the properties of some neighborhood of that pixel, rather than by, say, the position of the pixel. This assumption means that we can construct a model for the texture outside the boundaries of our example region, based on the properties of our example region. The assumption often applies to natural textures over a reasonable range of scales. For example, the stripes on a zebra's back are homogenous, but remember that those on its back are vertical and those on its legs, horizontal. We now use the example texture to obtain the probability model for the synthesized texture in various ways.

10.3.2 Synthesis by Matching Histograms of Filter Responses

If two homogenous texture samples are drawn from the same probability model, then (if the samples are big enough) histograms of the outputs of various filters applied to the samples will be the same. Heeger and Bergen use this observation to synthesize a texture using the following strategy: take a noise image and adjust it

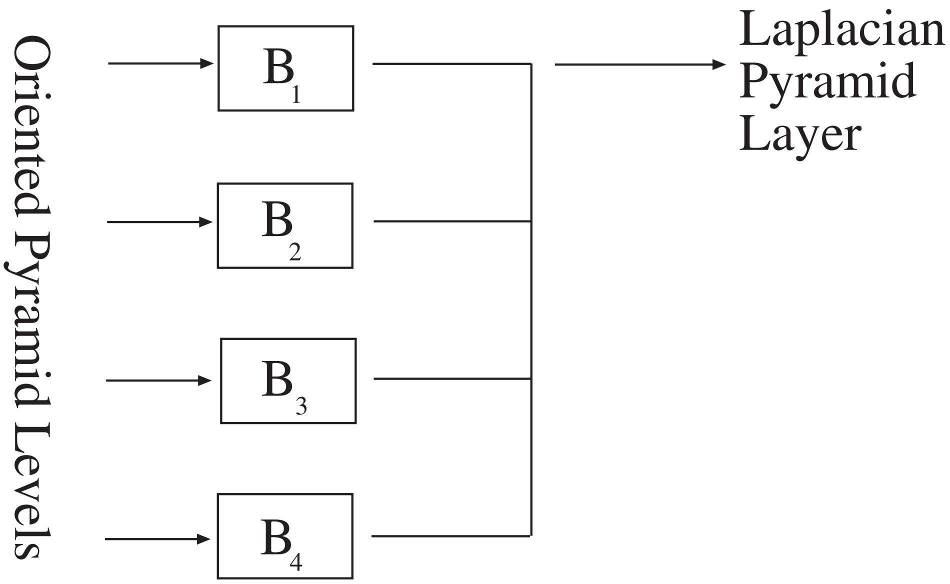


Figure 10.12. In the oriented pyramid, synthesis is possible by refiltering the layers and then adding them, as this schematic indicates. This property is obtained by appropriate choice of filters.

until the histogram of responses of various filters on that noise image looks like the histogram of responses of these filters on the texture sample.

Using an arbitrary set of filters is likely to be inefficient; we can avoid this problem by using an oriented pyramid. As we have seen, each orientation of each layer represents the response of an oriented filter at a particular scale, so the whole pyramid represents the response of a large number of different filters[1].

If we represent texture samples as oriented pyramids, we can adjust the pyramid corresponding to the image to be synthesized, and then synthesize the image from the pyramid, using the methods of section 10.2. We will adjust each layer separately, and then synthesize an image. The details of the process for adjusting the layers is given in section 4; for the moment, we will assume that this process works, and discuss what we do with it.

Once we have obtained an image from the adjusted pyramid, we form a pyramid from that image (the two pyramids will not, in general, be the same, because we've assumed, incorrectly, that the layers are independent). In particular, we are not guaranteed that each layer in the new pyramid has the histogram we want it to. If the layer histograms are not satisfactory, we readjust the layers, resynthesize the image, and iterate. While convergence is not guaranteed, in practice the process appears to converge.

---

[1] The reason this is efficient is that we have thrown away redundant information in subsampling the images to get the coarser scale layers.

```
make a working image from noise
match the working image histogram to the example image histogram
make a pyramid pe from the example image

until convergence
  Make a pyramid pw from the working image
  For each layer in the two pyramids
    match the histogram of pw’s layer to that of pe’s layer
  end
  synthesize the working image from the pyramid pw
end
```

Algorithm 10.3: *Iterative texture synthesis using histogram equalisation applied to an oriented pyramid*

The overall technique looks like algorithm 3. This algorithm yields quite good results on a substantial variety of textures, as figure 10.13 indicates. It is inclined to fail when there are conditional relations in the texture that are important — for example, in figure 10.14, the method has been unable to capture the fact that the spots on the coral lie in stripes. This problem results from the assumption that the histogram at each spatial frequency and orientation is independent of that at every other.

Histogram Equalization

We have two images — which might be layers from the oriented pyramid — and we should like to adjust image two so that it has the same histogram as image one. The process is known as **histogram equalization**. Histogram equalization is easiest for images that are continuous functions. In this case, we record for each value of the image the percentage of the image that takes the value less than or equal to this one-this record is known as the **cumulative histogram**. The cumulative histogram is a continuous, monotonically increasing function that maps the range of the image to the unit interval. Because it is continuous and monotonically increasing, the inverse exists. The inverse of the cumulative histogram takes a percentage — say 25 % — and gives the image value v such that the given percentage of the image has value less than or equal to v — i.e. 0.3, if 25% of the image has value less than or equal to 0.3.

The easiest way to describe histogram equalisation is slightly inefficient in space. We create a temporary image, image three. Now choose some value v from image two. The cumulative histogram of image two yields that p percent of the image

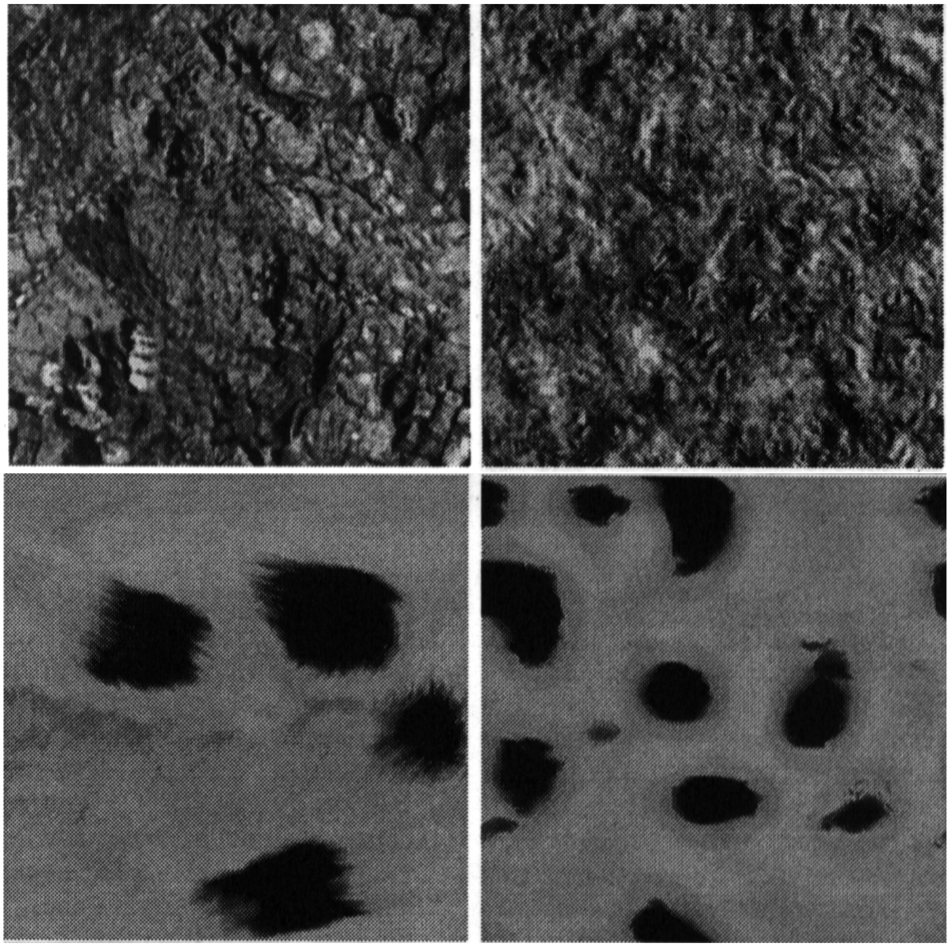


Figure 10.13. Examples of texture synthesis by histogram equalisation. On the left, the example textures and on the right, the synthesized textures. For the top example, the method is unequivocally successful. For the bottom example, the method has captured the spottiness of the texture but has rather more (and smaller) spots than one might expect. *figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 3, in the fervent hope, etc.*

has value less than v. Now apply the inverse cumulative histogram of image one to p, yielding a new value v' for v. Wherever image two has the value v, insert the value v' in image three. If this is done for every value, image three will have the same histogram as image one. This is because, for any value in image three, the percentage of image three that has that value is the same as the percentage of image one that has that value. In fact, image three isn't necessary, as we can transform

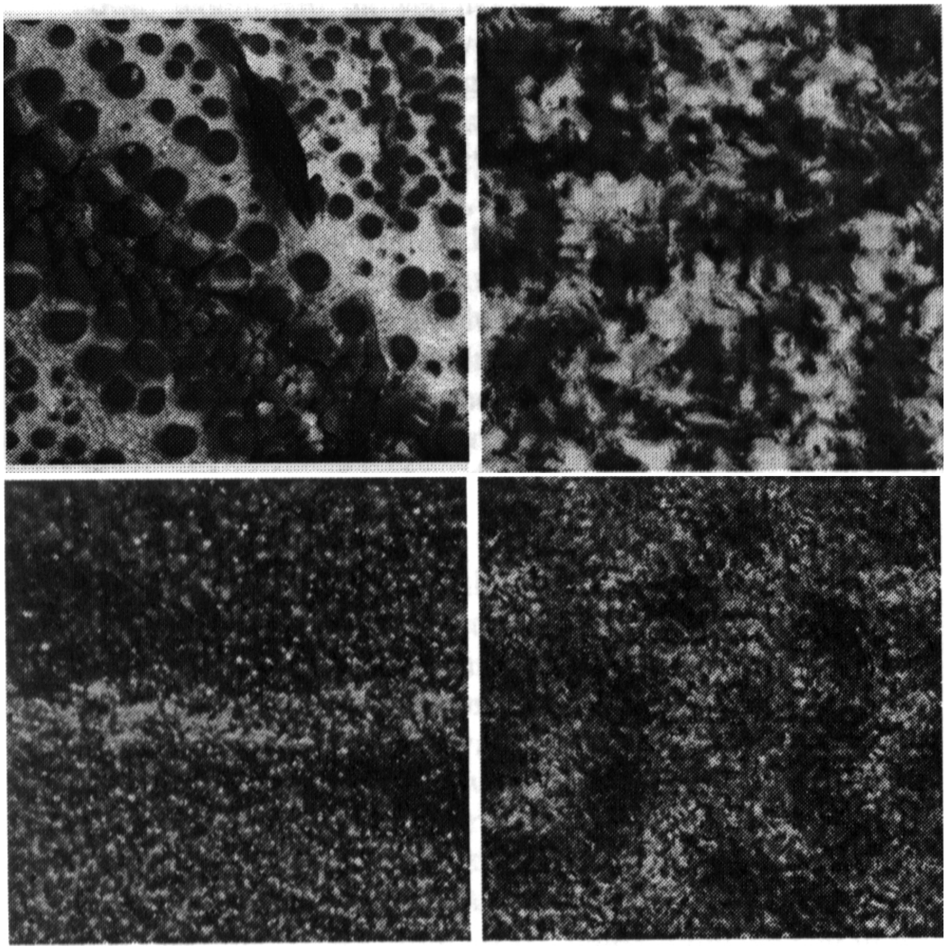


Figure 10.14. Examples of texture synthesis by histogram equalisation failing. The left column shows example textures, and the right hand column shows synthesized textures. The main phenomenon that causes failure is that, for most natural textures, the histogram of filter responses at different scales and orientations is not independent. In the case of the coral (top left), this independence assumption suppresses the fact that the small spots on the coral lie in a straight line. *figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 8, in the fervent hope, etc.*, *figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 7, in the fervent hope, etc.*

image two in place, yielding algorithm 4.

Things are slightly more difficult for discrete images and images that take discrete values. For example, if image one is a binary image in which every pixel but

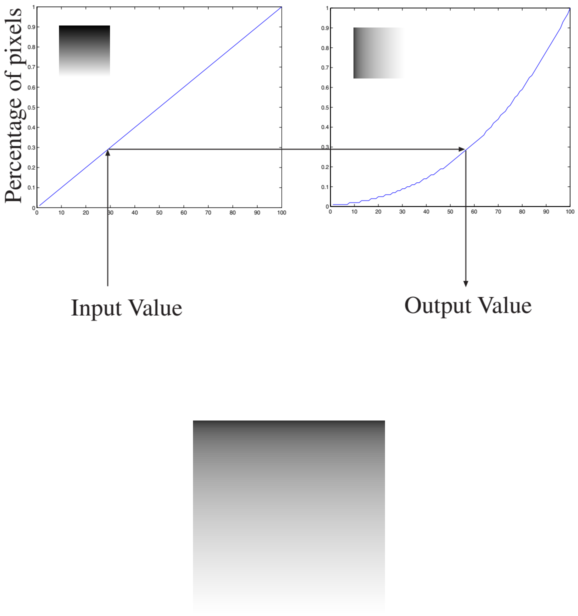


Figure 10.15. Histogram equalization uses cumulative histograms to map the grey levels of one image so that it has the same histogram as another image. The figure at the top shows two cumulative histograms, with the relevant images inset in the graphs. To transform the left image so that it has the same histogram as the right image, we take a value from the left image, read off the percentage from the cumulative histogram of that image, and obtain a new value for that grey level from the inverse cumulative histogram of the right image. The image on the left is a linear ramp (it looks non-linear because the relationship between brightness and lightness is not linear); the image on the right is a cube root ramp. The result — the linear ramp, with grey levels remapped so that it has the same histogram as the cube root ramp — is shown on the bottom row.

one is black, and image two is a binary image in which half the pixels are white, some but not all of the white pixels in image two will need to be mapped to black — but which ones should we choose? usually the choice is made uniformly and at random.

10.3.3 Synthesis by Sampling Conditional Densities of Filter Responses

A very successful algorithm due to DeBonet retains the idea of synthesizing a texture by coming up with an image pyramid that looks like the pyramid associated with an example texture. However, this approach does not assume that the layers are independent, as the previous algorithm did.

For each location in a given layer of the pyramid, there are a set of locations in

```
form the cumulative histogram c1(v) for image 1
form the cumulative histogram c2(v) for image 2
form ic1(p), the inverse of c1(v)

for every value v in image 2, going from smallest to largest
  obtain a new value vnew=ic1(c2(v))
  replace the value v in image 2 with vnew
end
```

Algorithm 10.4: *Histogram Equalization*

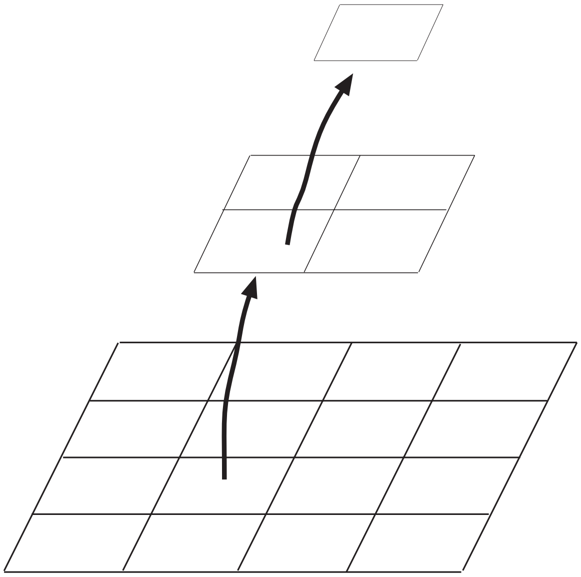


Figure 10.16. The values of pixels at coarse scales in a pyramid are a function of the values in the finer scale layers. We associate a parent structure with each pixel, which consists of the values of pixels at coarse scales which are used to predict our pixel's value in the Laplacian pyramid, as indicated in this schematic drawing. This parent structure contains information about the structure of the image around our pixel for a variety of differently sized neighbourhoods.

coarser scale layers associated with it by the sampling process (as in figure 10.16). The set of values of in these locations is called the **parent structure** of the location.

We can use this parent structure for synthesis. Firstly, let us make the coarsest scale in the new pyramid the same as the coarsest scale — say the m'th level —

```
make a pyramid pe from the example image
make an empty pyramid pw, corresponding to the image to
be synthesized

set the coarsest scale layer of pw to be the same as the
coarsest scale level of pe; if pw is bigger than pe, then
replicate copies of pe to fill it

for each other layer l of pe, going from coarsest to finest
  for each element e of the layer

    obtain all elements with
    the same parent structure

    choose one of this collection uniformly at random

    insert the value of this element into e

  end
end

synthesize the texture image from the pyramid pw
```

Algorithm 10.5: *Texture Synthesis using Conditional Histograms*

in the example pyramid. Now choose a location to be synthesized in the $m-1$'th level of the pyramid. We know the parent structure of this location, so we can go to the example pyramid and collect all values in the corresponding level that have a similar parent structure. This collection forms a probability model for the values for our location, conditioned on the parent structure that we observed. If we choose an element from this collection uniformly and at random, the values at the m'th level and at the $m-1$'th level of the pyramid being synthesized have the same *joint* histogram as the corresponding layers in the example pyramid.

This is easiest to see if we think of histograms as a representation of a probability distribution. The joint histogram is a representation of the joint probability distribution for values at the two scales. From section **??**, this joint distribution is the product of a marginal distribution on the values at the m'th level with the conditional distribution on values at the $m-1$'th level, *conditioned* on the value at the m'th level.

The m'th level layers must have the same histograms (that is, the same marginal

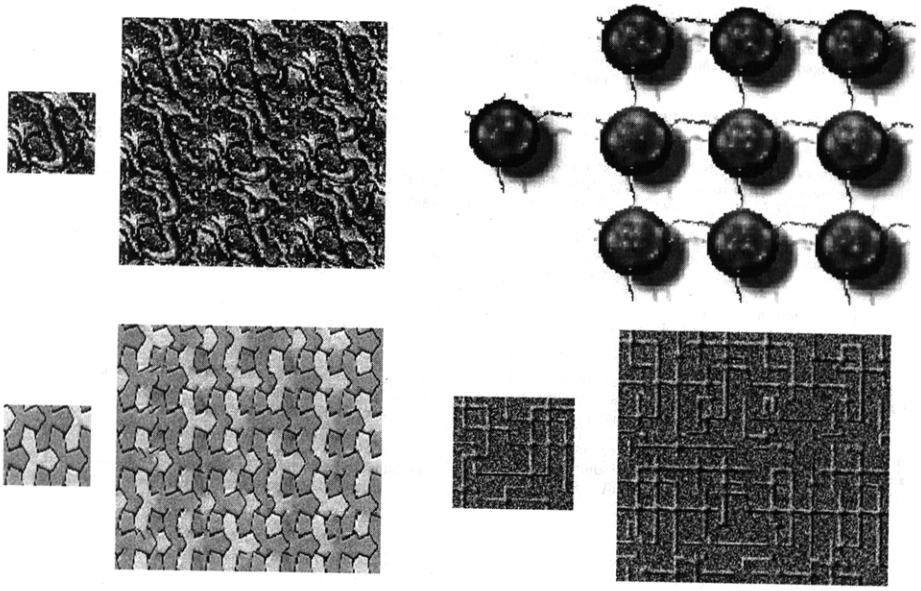


Figure 10.17. Four examples of textures synthesized using De Bonet's algorithm (algorithm 5). In each case, the example texture is the small block on the left, and the synthesized texture is the larger image block on the right. Note that the method has captured apparent periodic structure in the textures; in the case of the blob with wires (top right), it has succeeded in joining up wires. This is because the method can capture larger scale structure in a texture in greater detail, by not assuming that responses at each level of the pyramid are independent. *figure from De Bonet, Multiresolution Sampling Procedure for Analysis and Synthesis of Image Textures, p figure 10, in the fervent hope, etc.*

distributions). The sampling procedure for the $m-1$'th layer means that a histogram of the pixels in the $m-1$'th layer whose parents have some fixed value will be the same for the pyramid being synthesized as for the example pyramid. This histogram — which is sometimes called a **conditional histogram** — is a representation of the conditional distribution on values at the $m-1$'th level, *conditioned* on the value at the m'th level.

Nothing special is required to synthesize a third (or any other) layer. For any location in the third layer, the parent structure involves values from the coarsest and the next to coarsest scale. To obtain a value for a location, we collect every element from the corresponding layer in the example pyramid with the same parent structure, and choose from a uniformly and at random from this collection. The fourth, fifth and other layers follow from exactly the same approach. Furthermore, the joint histogram of all these layers in the synthesized pyramid will be the same

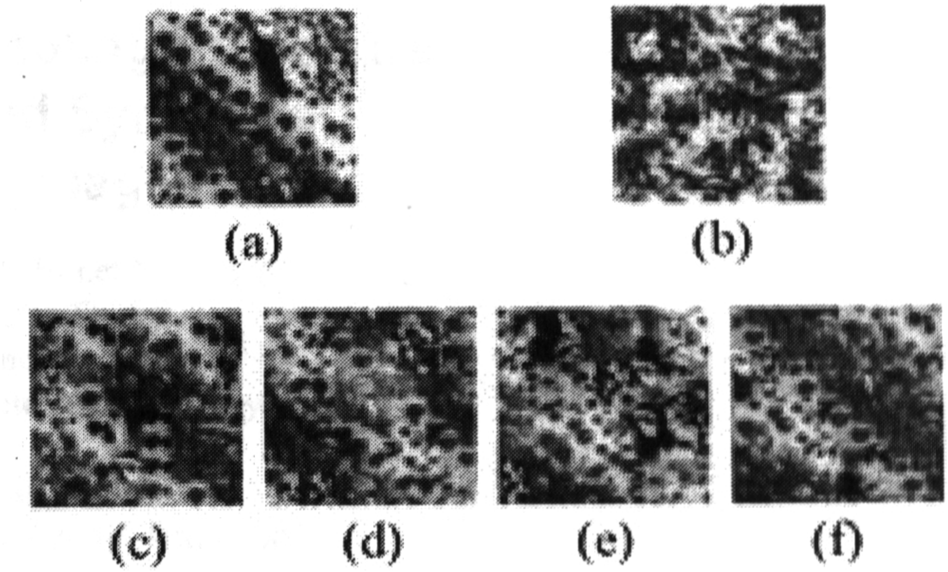


Figure 10.18. Figure 10.13 showed texture synthesis by histogram equalisation failing on the coral texture example shown on the top left here, because the independence assumption suppresses the fact that the small spots on the coral lie in a straight line. The texture synthesized by histogram equalization is shown on the top right. The bottom row shows textures synthesized using algorithm 5, which doesn't require an independence assumption. These textures appear to have the same structure as the example. *figure from De Bonet, Multiresolution Sampling Procedure for Analysis and Synthesis of Image Textures, p figure 14, in the fervent hope, etc.*

as that for the example pyramid, using the same argument as above.

There are two important details to address before we have a usable algorithm.

- Firstly, what does it mean for parent structures to be the same? In practice, it is sufficient to regard the parent structures as vectors and require that they are close together — an appropriate distance threshold should be set by experiment.
- Secondly, how do we obtain all pixels with the same parent structure as a given location? one strategy is to search all locations in the example image for every pixel value we wish to synthesize, but this is crude and expensive. We explore alternate strategies in exercise **??**.

10.3.4 Synthesis by Sampling Local Models

As Efros points out, it isn't essential to use an oriented pyramid to build a probability model. Instead, the example image itself supplies a probability model. Assume for the moment that we have every pixel in the synthesized image, except one. To obtain a probability model for the value of that pixel, we could match a neighborhood of the pixel to the example image. Every matching neighborhood in the example image has a possible value for the pixel of interest. This collection of values is a conditional histogram for the pixel of interest. By drawing a sample uniformly and at random from this collection, we obtain the value that is consistent with the example image. Section **??** describes the details of the matching process.

```
Choose a small square of pixels at random from the example image
Insert this square of values into the image to be synthesized

until each location in the image to be synthesized has a value
  For each unsynthesized location on
  the boundary of the block of synthesized values
    Match the neighborhood of this location to the
    example image, ignoring unsynthesized
    locations in computing the matching score

    Choose a value for this location uniformly and at random
    from the set of values of the corresponding locations in the
    matching neighborhoods
  end
end
```

Algorithm 10.6: *Non-parametric Texture Synthesis*

Generally, we need to synthesize more than just one pixel. Usually, the values of some pixels in the neighborhood of the pixel to be synthesized are not known — these pixels need to be synthesized too. One way to obtain a collection of examples for the pixel of interest is to count only the known values in computing the sum of squared differences, and to adjust the threshold pro rata. The synthesis process can be started by choosing a block of pixels at random from the example image, yielding algorithm 6.

Matching Image Neighbourhoods

Efros uses a square neighborhood, centered at the pixel of interest. The size of the neighborhood is a parameter that significantly affects the appearance of the synthesized image (see figure 10.20). The similarity between two image neighbourhoods can be measured by forming the sum of squared differences of corresponding pixel values. This value is small when the neighbourhoods are similar, and large when they are different (it is essentially the length of the difference vector). Of course, the

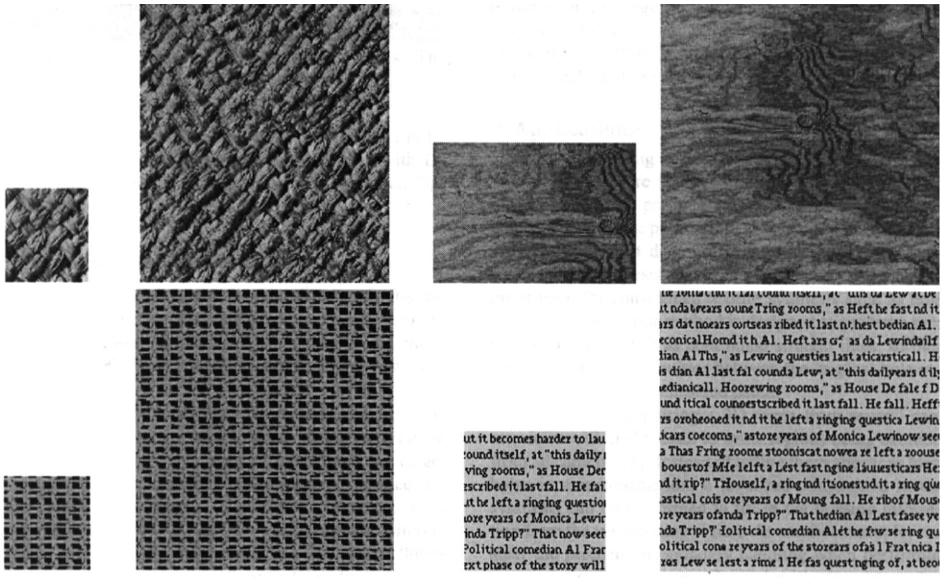


Figure 10.19. Efros' texture synthesis algorithm (algorithm 6) matches neighbourhoods of the image being synthesized to the example image, and then chooses at random amongst the possible values reported by matching neighbourhoods. This means that the algorithm can reproduce complex spatial structures, as these examples indicate. The small block on the left is the example texture; the algorithm synthesizes the block on the right. Note that the synthesized text looks like text; it appears to be constructed of words of varying lengths that are spaced like text; and each word looks as though it is composed of letters (though this illusion fails as one looks closely). *figure from Efros, Texture Synthesis by Non-parametric sampling, p. figure 3, in the fervent hope, etc.*

value of the pixel to be synthesized is not counted in the sum of squared differences.

The set of possible values for the pixel of interest comes from any neighborhood of the example image whose sum of squared differences with the neighborhood of interest is smaller than some threshold. Other choices of neighbourhood, and of matching criterion, might work well; little is known about what is best.

10.4 Shape from Texture: Planes and Isotropy

A patch of texture of viewed frontally looks very different from a same patch viewed at a glancing angle, because foreshortening appears to make the texture elements smaller, and move them closer together. This means that, if a surface is covered with the same texture, we should be able to tell elements that are frontal from those that are viewed at a glancing angle. By doing so, we can recover the shape of the surface (figure 10.21).

To construct useful algorithms, we need to be crisp about what it means for

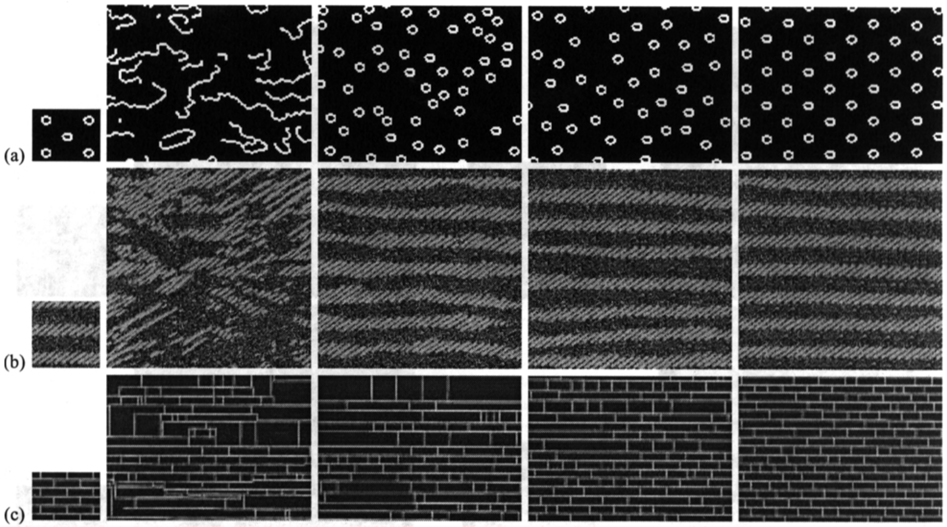


Figure 10.20. The size of the image neighbourhood to be matched makes a significant difference in algorithm 6. In the figure, the textures at the right are synthesized from the small blocks on the left, using neighbourhoods that are increasingly large as one moves to the right. If very small neighbourhoods are matched, then the algorithm cannot capture large scale effects easily. For example, in the case of the spotty texture, if the neighbourhood is too small to capture the spot structure (and so sees only pieces of curve), the algorithm synthesizes a texture consisting of curve segments. As the neighbourhood gets larger, the algorithm can capture the spot structure, but not the even spacing. With very large neighbourhoods, the spacing is captured as well. *figure from Efros, Texture Synthesis by Non-parametric sampling, p. figure 2, in the fervent hope, etc.*

a texture to be the same. In the first case, let us assume that we are looking at textured planes. There are two useful notions of similarity for this case. We discussed homogenous textures above (section 10.3.1); an **isotropic** texture is one where the probability of encountering a texture element does not depend on the orientation of that element. This means that a probability model for an isotropic texture need not depend on the orientation of the coordinate system on the textured plane. We will confine our discussion to the case of an orthographic camera. If the camera is not orthographic, the arguments we use will go through, but require substantially more work and more notation. Derivations for other cases appear in [].

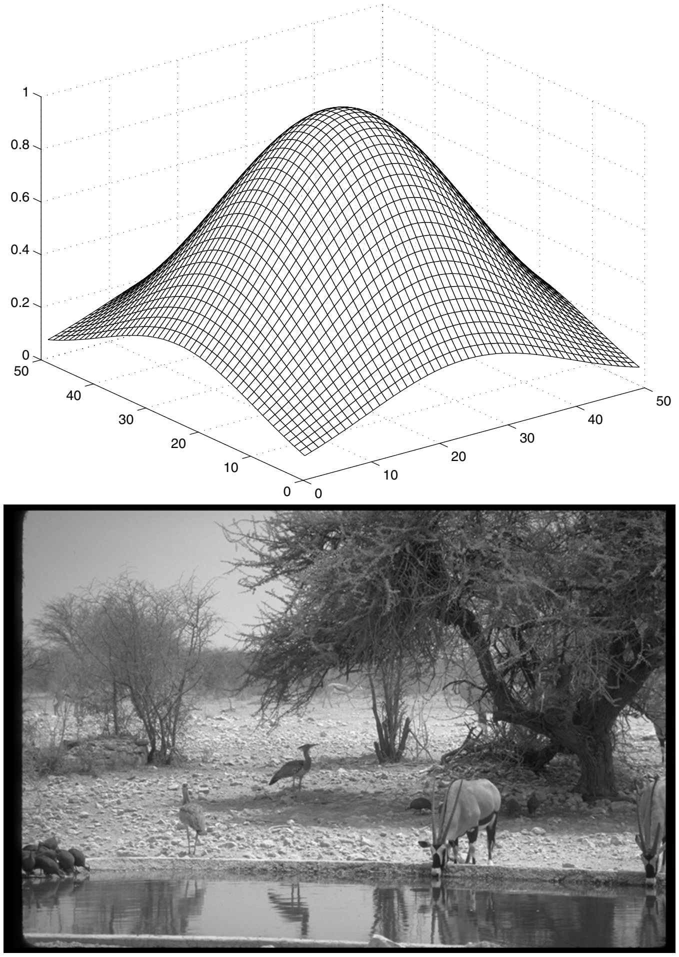


Figure 10.21. Humans obtain information about the shape of surfaces in space from the appearance of the texture on the surface. The figure on the left shows one common use for this effect — away from the contour regions, our only source of information about the surface depicted is the distortion of the texture on the surface. On the right, the texture of the stones gives a clear sense of the orientation of the (roughly) plane surface leading up to the waterhole. *figure from the Calphotos collection, number. 0127, in the fervent hope, etc.*

10.4.1 Recovering the Orientation of a Plane from an Isotropic Texture

Now assume that we are viewing a single textured plane in an orthographic camera. Because the camera is orthographic , there is no way to measure the depth to the plane. However, we can think about the orientation of the plane. Let us work in terms of the camera coordinate system. We need to know firstly, the angle between the normal of the textured plane and the viewing direction — sometimes called the **slant** — and secondly, the angle the projected normal makes in the camera coordinate system — sometimes called the **tilt** (figure 10.22). In an image of a plane, there is a **tilt direction** — the direction in the plane parallel to the projected normal.

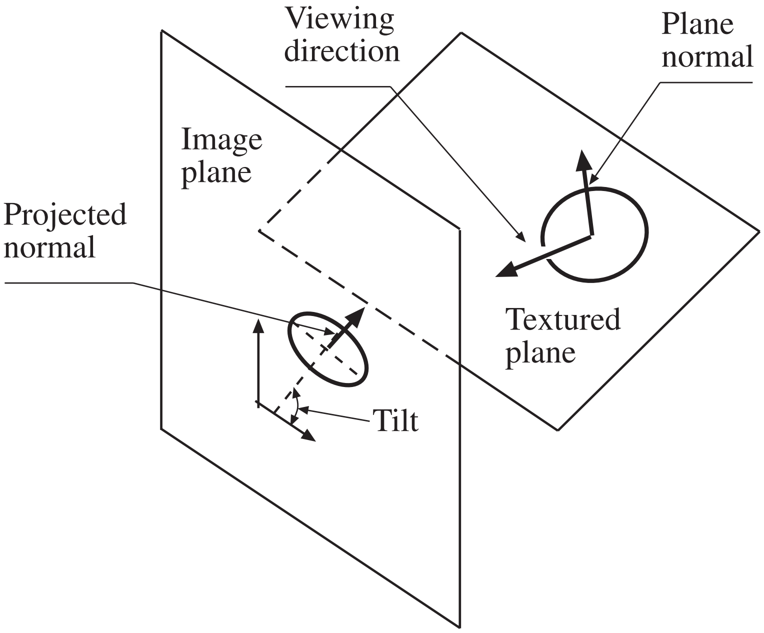


Figure 10.22. The orientation of a plane with respect to the camera plane can be given by the slant — which is the angle between the normal of the textured plane and the viewing direction — and the tilt — which is the angle the projected normal makes with the camera coordinate system. The figure illustrates the tilt, and shows a circle projecting to an ellipse.

If we assume that the texture is isotropic, both slant and tilt can be read from the image. We could synthesize an orthographic view of a textured plane by first rotating the coordinate system by the tilt and then secondly contracting along one coordinate direction by the cosine of the slant — call this process a **viewing transformation**. The easiest way to see this is to assume that the texture consists of a set of circles, scattered about the plane. In an orthographic view, these circles will project to ellipses, whose minor axes will give the tilt, and whose aspect ratios will give the slant (see exercise **??** and figure 10.22).

The process of contraction interferes with the isotropy of the texture, because elements that point along the contracted direction get shorter. Furthermore, elements that have a component along the contracted direction have that component shrunk. This yields a strategy for determining the orientation of the plane: find a viewing transformation that turns the image texture into an isotropic texture, and recover the slant and tilt from that viewing transformation.

There are variety of ways to find this viewing transformation. One natural strategy is to use the energy output of a set of oriented filters. This is the squared response, summed over the image. For an isotropic texture, we would expect the energy output to be the same for each orientation at any given scale, because the probability of encountering a pattern does not depend on its orientation. Thus, a measure of isotropy is the standard deviation of the energy output as a function of orientation. We could sum this measure over scales, perhaps weighting the measure by the total energy in the scale. The smaller the measure, the more isotropic the texture. We now find the inverse viewing transformation that makes the image looks most isotropic by this measure, using standard methods from optimization.

Notice that this approach immediately extends to perspective projection, spherical projection, and other types of viewing transformation. We simply have to search over a larger family of transformations for the transformation that makes the image texture look most isotropic. One does need to be careful, however. For example, scaling an isotropic texture will lead to another isotropic texture, meaning that it isn't possible to recover a scaling parameter, and it's a bad idea to try. Notice also that it isn't possible to recover the configuration of a plane from an orthographic image if one assumes the plane is homogenous — an affine transformation of a homogenous texture is homogenous.

The main difficulty with using an assumption of isotropy to recover the orientation of a plane is that there are very few isotropic textures in the world. Curved surfaces have a richer geometric structure — basically, the texture at different points is distorted in different ways — and we can recover that structure with more realistic assumptions.

10.4.2 Recovering the Orientation of a Plane from an Homogeneity Assumption

It isn't possible to recover the orientation of a plane in an orthographic view by assuming that the texture is homogeneous (the definition is in section 10.3.1). This is because the viewing transformation takes one homogeneous texture into another homogeneous texture. However, if we make some other assumptions about the structure of the texture, it becomes possible. One possible assumption is that the texture is a **homogenous marked Poisson point process**. This is a special case of the Poisson point process described in section 9.5.1; in particular, the texture is obtained by (1) marking points on the plane with a homogenous Poisson point process and then (2) dropping a texture element (a "mark") at each point, with the choice of element and orientation being random with some fixed distribution.

Assume that we can identify each texture element. Now recall that the core property of a homogenous Poisson point process is that the expected number of points in a set is proportional to the area of that set. Consider a set of rectangles *in the image* oriented by the slant-tilt coordinate system: a rectangle that is long in the slant direction and short in the tilt direction will contain more texture elements than a rectangle that is long in the tilt direction and short in the slant direction. This is because the slant direction is foreshortened — so that image length in this direction is shorter than length on the plane in this direction — but the tilt direction is not. However, the foreshortening does not affect the count of texture elements. These observations mean that we can obtain the slant and tilt direction of a plane textured according to our model by searching over plane orientations to find one that makes the back-projected texture most uniform in space.

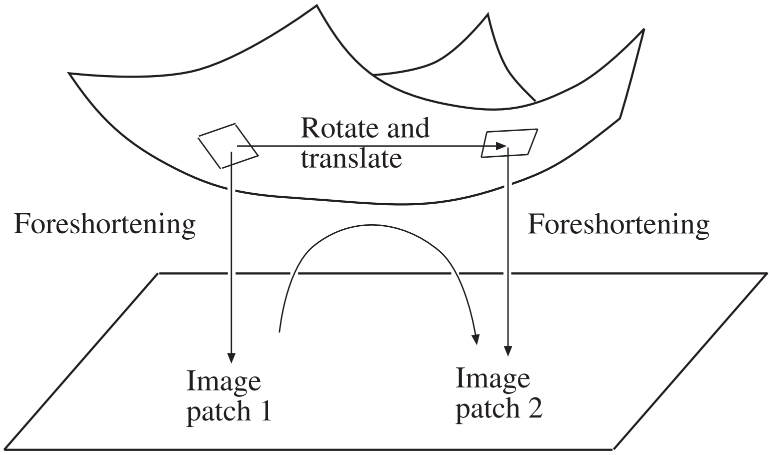


Figure 10.23. If the texture on a surface is homogenous, then the texture at each point on the surface "looks like" the texture at other points. This means that the deformation of the texture in the image is a cue to surface geometry. In particular, the texture around one point in the image is related to the texture around another point by: mapping from the image to the surface, transforming on the surface, and then mapping back to the image. By keeping track of these transformations, we can reconstruct surfaces up to some ambiguity.

10.4.3 Shape from Texture for Curved Surfaces

If a homogenous texture lies on a curved surface, we can recover information about the differential geometry of that surface. The reasoning is as follows:

- We assume that texture is homogeneous. This means that, if we know the configuration of one of the tangent planes on the surface, then we know what the texture looks like frontally.

- Now we assume that we know the configuration of one of the tangent planes.

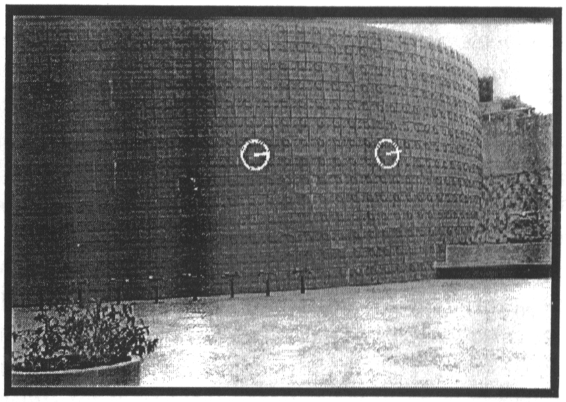


Figure 10.24. A textured cylinder, with a representation of surface normals obtained by reasoning about relative distortion from point to point. *figure from the Malik and Rosenholtz, Computing Local Surface Orientation and Shape from Texture for Curved Surfaces, p.161, in the fervent hope, etc.*

- Now we can reconstruct other tangent planes — possibly every other tangent plane — from this information, because we know the rule by which the texture foreshortens.

Of course, we don't know the configuration of any of the tangent planes, so we need to reason about relative configurations. The texture distorts from place to place in the image, because it undergoes different projections into the image: we keep track of those distortions, and use them to reason about the shape of the surface (figures 10.23 and 10.24). Shape from texture for curved surfaces tends to require some technical geometry, however, and we will pursue it no further.

10.5 Notes

We have aggressively compressed the texture literature in this chapter. Over the years, there have been a wide variety of techniques for representing image textures, typically looking at the statistics of how patterns lie with respect to one another. The disagreements are in how a pattern should be described, and what statistics to look at. While it is a bit early to say that the approach that represents patterns using linear filters is correct, it is currently dominant, mainly because it is very easy to solve problems with this strategy. Readers who are seriously interested in texture will probably most resent our omission of the Markov Random Field model, a choice based on the amount of mathematics required to develop the model and the absence of satisfactory inference algorithms for MRF's. We refer the interested reader to [].

Texture synthesis exhausted us long before we could exhaust it. The most significant omission, apart from MRF's, is the work of Zhu and Mumford, which uses sophisticated entropy criteria to firstly choose filters by which to represent a texture and secondly construct probability models for that texture.

10.5.1 Shape from Texture

We have not handled shape from texture in any great detail, from a conviction that the literature is unsatisfactory. Local methods for curved surfaces currently require quite implausible assumptions about the rotational properties of the texture field to recover geometry (e.g. []). Furthermore, these methods recover both curvature and surface normal, thereby ignoring integrability. Most of what we have sketched applies only where the scale of the variation in the surface is much larger than the scale of variation in the texture — this should be a source of some unease, too. There is a great deal that can be done in this area, and the tools for understanding texture are now much better than they used to be.

Assignments

Exercises

1. The texture synthesis algorithm of section 10.3.3 needs to obtain parent structures in the example image that match the parent structure of a pixel to be synthesized. These could be obtained by blank search. An alternative is to use a hashing process. It is essential that every parent structure that could match a given structure is obtained by this hashing process. One strategy is to compute a hash key from the parent structure, and then look at nearby keys as well, to ensure that no matches are missed.
    - Describe how this strategy could work.
    - What savings could be obtained by using it?
2. Show that a circle appears as an ellipse in an orthographic view, and that the minor axis of this ellipse is the tilt direction. What is the aspect ratio of this ellipse?
3. We will study measuring the orientation of a plane in an orthographic view, given the texture consists of points laid down by a homogenous Poisson point process. Recall that one way to generate points according to such a process is to sample the x and y coordinate of the point uniformly and at random. We assume that the points from our process lie within a unit square.
    - Show that the probability that a point will land in a particular set is proportional to the area of that set.
    - Assume we partition the area into disjoint sets. Show that the number of points in each set has a multinomial probability distribution.

We will now use these observations to recover the orientation of the plane. We partition the *image texture* into a collection of disjoint sets.

- Show that the area of each set, *backprojected onto the textured plane*, is a function of the orientation of the plane.
- Use this function to suggest a method for obtaining the plane's orientation.

Programming Assignments

- **Texture synthesis - a:** Implement the texture synthesis algorithms of section 10.3.2 and of section 10.3.3. Use the steerable filter implementation available at `http://www.cis.upenn.edu/ eero/steerpyr.html` to construct steerable pyramid representations. Use your implementation to find examples where the independence assumption fails. Explain what is going on in these examples.

- **Texture synthesis - b:** Extend the algorithms of section 10.3.2 and of section 10.3.3 to use pyramids obtained using an analysis based on more orientations; you will need to ensure that you can do synthesis for the set of filters you choose. Does this make any difference in practice to (a) the quality of the texture synthesis or (b) the speed of the synthesis algorithm?

- **Texture synthesis - c:** Implement the non-parametric texture synthesis algorithm of section 10.3.4. Use your implementation to study:

  1. the effect of window size on the synthesized texture;
  2. the effect of window shape on the synthesized texture;
  3. the effect of the matching criterion on the synthesized texture (i.e. using weighted sum of squares instead of sum of squares, etc.).

Part IV

EARLY VISION: MULTIPLE IMAGES

Chapter 11

THE GEOMETRY OF MULTIPLE VIEWS

Despite the wealth of information contained in a photograph, the depth of a scene point along the corresponding projection ray is not directly accessible in a single image. With at least two pictures, on the other hand, depth can be measured through triangulation. This is of course one of the reasons why most animals have at least two eyes and/or move their head when looking for friend or foe, as well as the motivation for equipping autonomous robots with stereo or motion analysis systems. Before building such a program, we must understand how several views of the same scene constrain its three-dimensional structure as well as the corresponding camera configurations. This is the goal of this chapter.

In particular, we will elucidate the geometric and algebraic constraints that hold among two, three, or more views of the same scene. In the familiar setting of binocular stereo vision, we will show that the first image of any point must lie in the plane formed by its second image and the optical centers of the two cameras. This *epipolar constraint* can be represented algebraically by a 3×3 matrix called the *essential matrix* when the intrinsic parameters of the cameras are known, and the *fundamental matrix* otherwise. Three pictures of the same line will introduce a different constraint, namely that the intersection of the planes formed by their preimages be degenerate. Algebraically, this geometric relationship can be represented by a $3 \times 3 \times 3$ *trifocal tensor.* More images will introduce additional constraints, for example four projections of the same point will satisfy certain quadrilinear relations whose coefficients are captured by the *quadrifocal tensor*, etc. Remarkably, the equations satisfied by multiple pictures of the same scene feature can be set up without any knowledge of the cameras and the scene they observe, and a number of methods for estimating their parameters directly from image data will be presented in this chapter.

Computer vision is not the only scientific field concerned with the geometry of multiple views: the goal of photogrammetry, already mentioned in Chapter 5, is precisely to recover quantitative geometric information from multiple pictures. Applications of the epipolar and trifocal constraints to the classical photogrammetry problem of *transfer* (i.e., the prediction of the position of a point in an image given its position in a number of reference pictures) will be briefly discussed in this chapter, along with some examples. Many more applications in the domains of stereo and motion analysis will be presented in latter chapters.

11.1 Two Views

11.1.1 Epipolar Geometry

Consider the images p and p' of a point P observed by two cameras with optical centers O and O'. These five points all belong to the *epipolar plane* defined by the two intersecting rays OP and $O'P$ (Figure 11.1). In particular, the point p' lies on the line l' where this plane and the retina Π' of the second camera intersect. The line l' is the *epipolar line* associated with the point p, and it passes through the point e' where the *baseline* joining the optical centers O and O' intersects Π'. Likewise, the point p lies on the epipolar line l associated with the point p', and this line passes through the intersection e of the baseline with the plane Π.

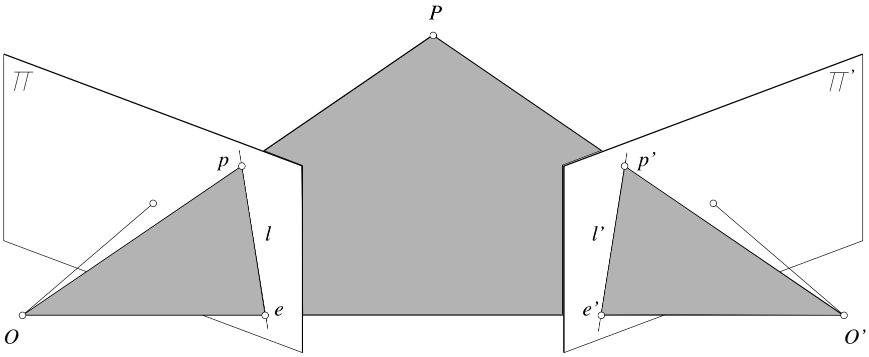


Figure 11.1. Epipolar geometry: the point P, the optical centers O and O' of the two cameras, and the two images p and p' of P all lie in the same plane.

The points e and e' are called the *epipoles* of the two cameras. The epipole e' is the (virtual) image of the optical center O of the first camera in the image observed by the second camera, and vice versa. As noted before, if p and p' are images of the same point, then p' must lie on the epipolar line associated with p. This *epipolar constraint* plays a fundamental role in stereo vision and motion analysis.

Let us assume for example that we know the intrinsic and extrinsic parameters of the two cameras of a stereo rig. We will see in Chapter 12 that the most difficult part of stereo data analysis is establishing correspondences between the two images, i.e., deciding which points in the right picture match the points in the left one. The epipolar constraint greatly limits the search for these correspondences: indeed, since we assume that the rig is calibrated, the coordinates of the point p completely determine the ray joining O and p, and thus the associated epipolar plane $OO'p$ and epipolar line. The search for matches can be restricted to this line instead of the whole image (Figure 11.2). In two-frame motion analysis on the other hand, each camera may be internally calibrated, but the rigid transformation separating the two camera coordinate systems is unknown. In this case, the epipolar geometry obviously constrains the set of possible motions. The next sections explore several variants of this situation.

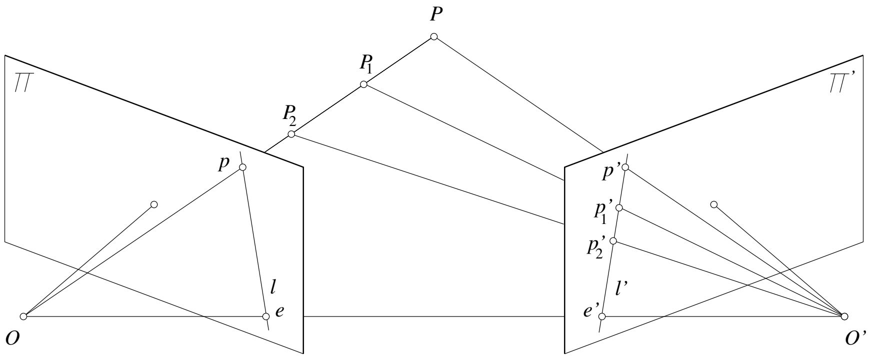


Figure 11.2. Epipolar constraint: given a calibrated stereo rig, the set of possible matches for the point p is constrained to lie on the associated epipolar line l'.

11.1.2 The Calibrated Case

Here we assume that the intrinsic parameters of each camera are known, so $\boldsymbol{p} = \hat{\boldsymbol{p}}$. Clearly, the epipolar constraint implies that the three vectors $\overrightarrow{Op}$, $\overrightarrow{O'p'}$, and $\overrightarrow{OO'}$ are coplanar. Equivalently, one of them must lie in the plane spanned by the other two, or

$$\overrightarrow{Op} \cdot [\overrightarrow{OO'} \times \overrightarrow{O'p'}] = 0.$$

We can rewrite this coordinate-independent equation in the coordinate frame associated to the first camera as

$$\boldsymbol{p} \cdot [\boldsymbol{t} \times (\mathcal{R}\boldsymbol{p}')], \tag{11.1.1}$$

where $\boldsymbol{p} = (u, v, 1)^T$ and $\boldsymbol{p}' = (u', v', 1)^T$ denote the homogenous image coordinate vectors of p and p', $\boldsymbol{t}$ is the coordinate vector of the translation $\overrightarrow{OO'}$ separating the two coordinate systems, and $\mathcal{R}$ is the rotation matrix such that a free vector with coordinates $\boldsymbol{w}'$ in the second coordinate system has coordinates $\mathcal{R}\boldsymbol{w}'$ in the first one (in this case the two projection matrices are given in the coordinate system attached to the first camera by $(\text{Id} \quad \mathbf{0})$ and $(\mathcal{R}^T, -\mathcal{R}^T\boldsymbol{t})$).

Equation (11.1.1) can finally be rewritten as

$$\boldsymbol{p}^T\mathcal{E}\boldsymbol{p}' = 0, \tag{11.1.2}$$

where $\mathcal{E} = [\boldsymbol{t}_\times]\mathcal{R}$, and $[\boldsymbol{a}_\times]$ denotes the skew-symmetric matrix such that $[\boldsymbol{a}_\times]\boldsymbol{x} = \boldsymbol{a}\times\boldsymbol{x}$ is the cross-product of the vectors $\boldsymbol{a}$ and $\boldsymbol{x}$. The matrix $\mathcal{E}$ is called the *essential matrix*, and it was first introduced by Longuet-Higgins [**?**]. Its nine coefficients are only defined up to scale, and they can be parameterized by the three degrees of freedom of the rotation matrix $\mathcal{R}$ and the two degrees of freedom defining the direction of the translation vector $\boldsymbol{t}$.

Note that $\mathcal{E}\boldsymbol{p}'$ can be interpreted as the coordinate vector representing the epipolar line associated with the point p' in the first image: indeed, an image line l can be defined by its equation $au + bv + c = 0$, where (u, v) denote the coordinates of a point on the line, (a, b) is the unit normal to the line, and $-c$ is the (signed) distance between the origin and l. Alternatively, we can define the line equation in terms of the homogeneous coordinate vector $\boldsymbol{p} = (u, v, 1)^T$ of a point on the line and the vector $\boldsymbol{l} = (a, b, c)^T$ by $\boldsymbol{l} \cdot \boldsymbol{p} = 0$, in which case the constraint $a^2 + b^2 = 1$ is relaxed since the equation holds independently of any scale change applied to $\boldsymbol{l}$. In this context, (11.1.2) expresses the fact that the point p lies on the epipolar line associated with the vector $\mathcal{E}\boldsymbol{p}'$. By symmetry, it is also clear that $\mathcal{E}^T\boldsymbol{p}$ is the coordinate vector representing the epipolar line associated with p in the second image.

It is obvious that essential matrices are singular since $\boldsymbol{t}$ is parallel to the coordinate vector $\boldsymbol{e}$ of the left epipole, so that $\mathcal{E}^T\boldsymbol{e} = -\mathcal{R}^T[\boldsymbol{t}_\times]\boldsymbol{e} = 0$. Likewise, it is easy to show that $\boldsymbol{e}'$ is a zero eigenvector of $\mathcal{E}$. As shown by Huang and Faugeras [**?**], essential matrices are in fact characterized by the fact that they are singular with two equal non-zero singular values (see exercises).

11.1.3 Small Motions

Let us now turn our attention to *infinitesimal* displacements. We consider a moving camera with translational velocity $\boldsymbol{v}$ and rotational velocity $\boldsymbol{\omega}$ and rewrite (11.1.2) for two frames separated by a small time interval δt. Let us denote by $\dot{\boldsymbol{p}} = (\dot{u}, \dot{v}, 0)^T$ the velocity of the point p, or *motion field*. Using the exponential representation of rotations,[1] we have (to first order):

$$\begin{cases} \boldsymbol{t} = \delta t\, \boldsymbol{v}, \\ \mathcal{R} = \mathrm{Id} + \delta t\, [\boldsymbol{\omega}_\times], \\ \boldsymbol{p}' = \boldsymbol{p} + \delta t\, \dot{\boldsymbol{p}}. \end{cases}$$

Substituting in (11.1.2) yields

$$\boldsymbol{p}^T[\boldsymbol{v}_\times](\mathrm{Id} + \delta t\, [\boldsymbol{\omega}_\times])(\boldsymbol{p} + \delta t\, \dot{\boldsymbol{p}}) = 0,$$

and neglecting all terms of order two or greater in δt yields:

$$\boldsymbol{p}^T([\boldsymbol{v}_\times][\boldsymbol{\omega}_\times])\boldsymbol{p} - (\boldsymbol{p} \times \dot{\boldsymbol{p}}) \cdot \boldsymbol{v} = 0. \tag{11.1.3}$$

---

[1] The matrix associated with the rotation whose axis is the unit vector $\boldsymbol{a}$ and whose angle is θ can be shown to be equal to $e^{\theta[\boldsymbol{a}_\times]} \overset{\text{def}}{=} \sum_{i=0}^{+\infty} \frac{1}{i!}(\theta[\boldsymbol{a}_\times])^i$.

Equation (11.1.3) is simply the instantaneous form of the Longuet-Higgins relation (11.1.2) which captures the epipolar geometry in the discrete case. Note that in the case of pure translation we have $\boldsymbol{\omega} = 0$, thus $(\boldsymbol{p} \times \dot{\boldsymbol{p}}) \cdot \boldsymbol{v} = 0$. In other words, the three vectors $\boldsymbol{p} = \overrightarrow{op}$, $\dot{\boldsymbol{p}}$ and $\boldsymbol{v}$ must be coplanar. If e denotes the infinitesimal epipole, or *focus of expansion*, i.e., the point where the line passing through the optical center and parallel to the velocity vector $\boldsymbol{v}$ pierces the image plane, we obtain the well known result that the motion field points toward the focus of expansion under pure translational motion (Figure 11.3).

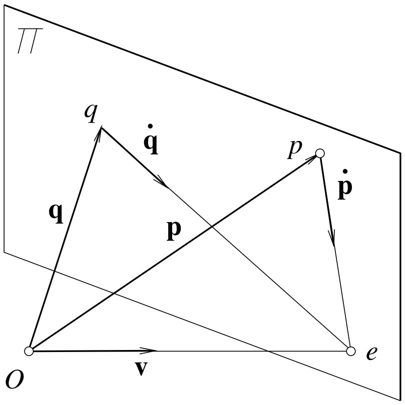


Figure 11.3. Focus of expansion: under pure translation, the motion field at every point in the image points toward the focus of expansion.

11.1.4 The Uncalibrated Case

The Longuet-Higgins relation holds for *internally calibrated* cameras, whose intrinsic parameters are known so that image positions can be expressed in normalized coordinates. When these parameters are unknown (*uncalibrated* cameras), we can write $\boldsymbol{p} = \mathcal{K}\hat{\boldsymbol{p}}$ and $\boldsymbol{p}' = \mathcal{K}'\hat{\boldsymbol{p}}'$, where $\mathcal{K}$ and $\mathcal{K}'$ are 3×3 calibration matrices, and $\hat{\boldsymbol{p}}$ and $\hat{\boldsymbol{p}}'$ are normalized image coordinate vectors. The Longuet-Higgins relation holds for these vectors, and we obtain

$$\boldsymbol{p}^T \mathcal{F} \boldsymbol{p}' = 0, \tag{11.1.4}$$

where the matrix $\mathcal{F} = \mathcal{K}^{-T}\mathcal{E}\mathcal{K}'^{-1}$, called the *fundamental matrix*, is not, in general, an essential matrix.[2] It has again rank two, and the eigenvector of $\mathcal{F}$ (resp. $\mathcal{F}^T$) corresponding to its zero eigenvalue is as before the position $\boldsymbol{e}'$ (resp. $\boldsymbol{e}$) of the epipole. Note that $\mathcal{F}\boldsymbol{p}'$ (resp. $\mathcal{F}^T\boldsymbol{p}$) represents the epipolar line corresponding to the point $\boldsymbol{p}'$ (resp. $\boldsymbol{p}$) in the first (resp. second) image.

---

[2] Small motions can also be handled in the uncalibrated setting. In particular, Viéville and Faugeras [?] have derived an equation similar to (11.1.3) that characterizes the motion field of a camera with varying intrinsic parameters.

The rank-two constraint means that the fundamental matrix only admits seven independent parameters. Several choices of parameterization are possible, but the most natural one is in terms of the coordinate vectors $\boldsymbol{e} = (\alpha, \beta)^T$ and $\boldsymbol{e}' = (\alpha', \beta')^T$ of the two epipoles, and of the so-called *epipolar transformation* that maps one set of epipolar lines onto the other one. We will examine the properties of the epipolar transformation in Chapter 14 in the context of structure from motion. For the time being, let us just note (without proof) that this transformation is parameterized by four numbers a, b, c, d, and that the fundamental matrix can be written as

$$\mathcal{F} = \begin{pmatrix} b & a & -a\beta - b\alpha \\ -d & -c & c\beta + d\alpha \\ d\beta' - b\alpha' & c\beta' - a\alpha' & -c\beta\beta' - d\beta'\alpha + a\beta\alpha' + b\alpha\alpha' \end{pmatrix}. \tag{11.1.5}$$

11.1.5 Weak Calibration

As mentioned earlier, the essential matrix is defined up to scale by five independent parameters. It is therefore possible (at least in principle), to calculate it by writing (11.1.2) for five point correspondences. Likewise, the fundamental matrix is defined by seven independent coefficients (the parameters a, b, c, d in (11.1.5) are only defined up to scale) and can in principle be estimated from seven point correspondences. Methods for estimating the essential and fundamental matrices from a minimal number of parameters indeed exist (see [**?**] and Section 11.4), but they are far too involved to be described here. This section addresses the simpler problem of estimating the epipolar geometry from a redundant set of point correspondences between two images taken by cameras with unknown intrinsic parameters, a process known as *weak calibration*.

Note that the epipolar constraint (11.1.4) is a linear equation in the nine coefficients of the fundamental matrix $\mathcal{F}$:

$$(u, v, 1) \begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0 \Leftrightarrow (uu', uv', u, vu', vv', v, u', v', 1) \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0. \tag{11.1.6}$$

Since (11.1.6) is homogeneous in the coefficients of $\mathcal{F}$, we can for example set $F_{33} = 1$ and use eight point correspondences $p_i \leftrightarrow p'_i$ $(i = 1, .., 8)$ to set up an 8×8

system of non-homogeneous linear equations:

$$\begin{pmatrix} u_1u_1' & u_1v_1' & u_1 & v_1u_1' & v_1v_1' & v_1 & u_1' & v_1' \\ u_2u_2' & u_2v_2' & u_2 & v_2u_2' & v_2v_2' & v_2 & u_2' & v_2' \\ u_3u_3' & u_3v_3' & u_3 & v_3u_3' & v_3v_3' & v_3 & u_3' & v_3' \\ u_4u_4' & u_4v_4' & u_4 & v_4u_4' & v_4v_4' & v_4 & u_4' & v_4' \\ u_5u_5' & u_5v_5' & u_5 & v_5u_5' & v_5v_5' & v_5 & u_5' & v_5' \\ u_6u_6' & u_6v_6' & u_6 & v_6u_6' & v_6v_6' & v_6 & u_6' & v_6' \\ u_7u_7' & u_7v_7' & u_7 & v_7u_7' & v_7v_7' & v_7 & u_7' & v_7' \\ u_8u_8' & u_8v_8' & u_8 & v_8u_8' & v_8v_8' & v_8 & u_8' & v_8' \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \end{pmatrix} = - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix},$$

which is sufficient for estimating the fundamental matrix. This is the *eight-point* algorithm proposed by Longuet-Higgins [**?**] in the case of calibrated cameras. It will fail when the associated 8×8 matrix is singular. As shown in [**?**] and the exercises, this will only happen, however, when the eight points and the two optical centers lie on a quadric surface. Fortunately, this is quite unlikely since a quadric surface is completely determined by nine points, which means that there is in general no quadric that passes through ten arbitrary points.

When $n > 8$ correspondences are available, $\mathcal{F}$ can be estimated using linear least squares by minimizing

$$\sum_{i=1}^{n} (\boldsymbol{p}_i^T \mathcal{F} \boldsymbol{p}_i')^2 \tag{11.1.7}$$

with respect to the coefficients of $\mathcal{F}$ under the constraint that the vector formed by these coefficients has unit norm.

Note that both the eight-point algorithm and its least-squares version ignore the rank-two property of fundamental matrices.[3] To enforce this constraint, Luong *et al.* [**?**; **?**] have proposed to use the matrix $\mathcal{F}$ output by the eight-point algorithm as the basis for a two-step estimation process: first, use linear least squares to compute the position of the epipoles $\boldsymbol{e}$ and $\boldsymbol{e}'$ that minimize $|\mathcal{F}^T\boldsymbol{e}|^2$ and $|\mathcal{F}\boldsymbol{e}'|^2$; second, substitute the coordinates of these points in (11.1.5): this yields a linear parameterization of the fundamental matrix by the coefficients of the epipolar transformation, which can now be estimated by minimizing (11.1.7) via linear least squares.

The least-squares version of the eight-point algorithm minimizes the mean-squared *algebraic distance* associated with the epipolar constraint, i.e., the mean-squared value of $e(\boldsymbol{p}, \boldsymbol{p}') = \boldsymbol{p}^T\mathcal{F}\boldsymbol{p}'$ calculated over all point correspondences. This error function admits a geometric interpretation: in particular, we have

$$e(\boldsymbol{p}, \boldsymbol{p}') = \lambda \mathrm{d}(\boldsymbol{p}, \mathcal{F}\boldsymbol{p}') = \lambda' \mathrm{d}(\boldsymbol{p}', \mathcal{F}^T\boldsymbol{p}),$$

where $d(\boldsymbol{p}, \boldsymbol{l})$ denotes the (signed) Euclidean distance between the point $\boldsymbol{p}$ and the line $\boldsymbol{l}$, and $\mathcal{F}\boldsymbol{p}$ and $\mathcal{F}^T\boldsymbol{p}'$ are the epipolar lines associated with $\boldsymbol{p}$ and $\boldsymbol{p}'$. The scale factors λ and λ' are simply the norms of the vectors formed by the first two components of $\mathcal{F}\boldsymbol{p}'$ and $\mathcal{F}^T\boldsymbol{p}$, and their dependence on the pair of data points observed may bias the estimation process.

---

[3] The original algorithm proposed by Longuet-Higgins ignores the fact that essential matrices have rank two and two equal singular values as well.

It is of course possible to get rid of the scale factors and directly minimize the mean-squared distance between the image points and the corresponding epipolar lines, i.e.,

$$\sum_{i=1}^{n}[\mathrm{d}^2(\boldsymbol{p}_i, \mathcal{F}\boldsymbol{p}'_i) + \mathrm{d}^2(\boldsymbol{p}'_i, \mathcal{F}^T\boldsymbol{p}_i)].$$

This is a non-linear problem, regardless of the parameterization chosen for the fundamental matrix, but the minimization can be initialized with the result of the eight-point algorithm. This method was first proposed by Luong *et al.* [**?**], and it has been shown to provide results vastly superior to those obtained using the eight-point method.

Recently, Hartley [**?**] has proposed a normalized eight-point algorithm and has also reported excellent results. His approach is based on the observation that the poor performance of the plain eight-point method is due, for the most part, to poor numerical conditioning. Thus Hartley has proposed to translate and scale the data so it is centered at the origin and the average distance to the origin is $\sqrt{2}$ pixel. This dramatically improves the conditioning of the linear least-squares estimation process. Accordingly, his method is divided into four steps: first, transform the image coordinates using appropriate translation and scaling operators $\mathcal{T} : \boldsymbol{p}_i \rightarrow \tilde{\boldsymbol{p}}_i$ and $\mathcal{T}' : \boldsymbol{p}'_i \rightarrow \tilde{\boldsymbol{p}}'_i$. Second, use linear least squares to compute the matrix $\tilde{\mathcal{F}}$ minimizing

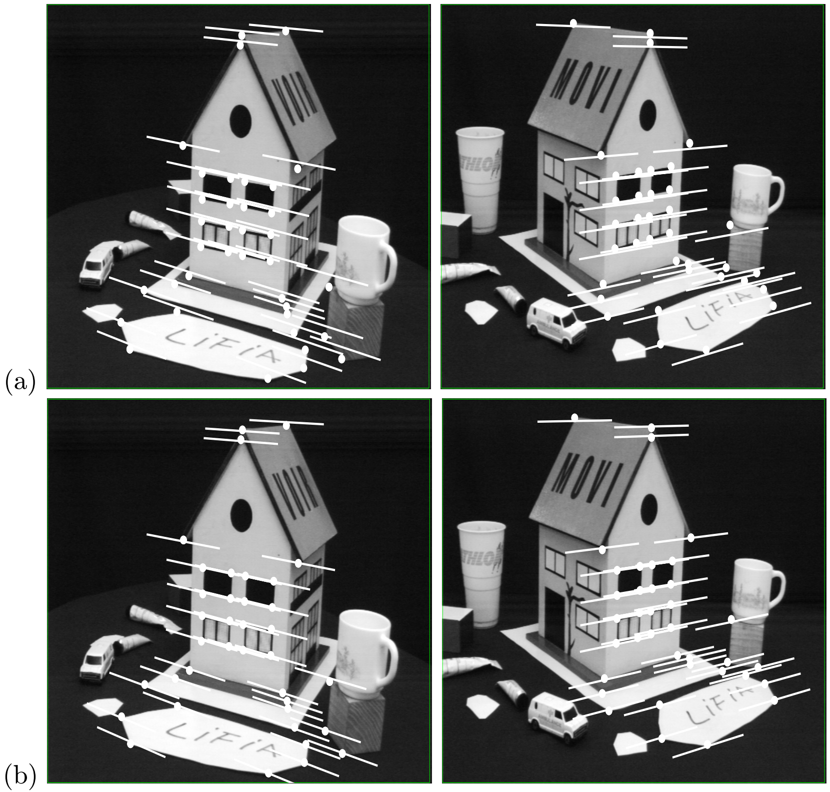
$$\sum_{i=1}^{n}(\tilde{\boldsymbol{p}}_i^T\tilde{\mathcal{F}}\tilde{\boldsymbol{p}}'_i)^2.$$

Third, enforce the rank-two constraint; this can be done using the two-step method of Luong *et al.* described earlier, but Hartley uses instead a technique, suggested by Tsai and Huang [**?**] in the calibrated case, which constructs the *singular value decomposition* $\tilde{\mathcal{F}} = \mathcal{U}\mathcal{S}\mathcal{V}^T$ of $\tilde{\mathcal{F}}$. Here, $\mathcal{S} = \mathrm{diag}(r, s, t)$ is a diagonal 3×3 matrix with entries $r \geq s \geq t$, and $\mathcal{U}, \mathcal{V}$ are orthogonal 3×3 matrices.[4] The rank-two matrix $\bar{\mathcal{F}}$ minimizing the Frobenius norm of $\tilde{\mathcal{F}} - \bar{\mathcal{F}}$ is simply $\bar{\mathcal{F}} = \mathcal{U}\mathrm{diag}(r, s, 0)\mathcal{V}^T$ [**?**]. Fourth, set $\mathcal{F} = \mathcal{T}^T\bar{\mathcal{F}}\mathcal{T}'$. This is the final estimate of the fundamental matrix.

Figure 11.4 shows weak calibration experiments using as input data a set of 37 point correspondences between two images of a toy house. The data points are shown in the figure as small discs, and the recovered epipolar lines are shown as short line segments. The top of the figure shows the output of the least-squares version of the plain eight-point algorithm, and the bottom part of the figure shows the results obtained using Hartley's variant of this method. As expected, the results are much better in the second case, and in fact extremely close to those obtained using the distance minimization criterion of Luong *et al.* [**?**; **?**].

---

[4] Singular value decomposition will be discussed in detail in Chapter 13.



	Linear Least Squares	[?]	[?]
Av. Dist. 1	2.33 pixels	0.92 pixel	0.86 pixel
Av. Dist. 2	2.18 pixels	0.85 pixel	0.80 pixel

Figure 11.4. Weak calibration experiments using 37 point correspondences between two images of a toy house. The figure shows the epipolar lines found by (a) the least-squares version of the 8-point algorithm, and (b) the "normalized" variant of this method proposed by Hartley [?]. Note for example the much larger error in (a) for the feature point close to the bottom of the mug. Quantitative comparisons are given in the table, where the average distances between the data points and corresponding epipolar lines are shown for both techniques as well as the non-linear distance minimization algorithm of Luong *et al.* [?].

11.2 Three Views

Let us now go back to the calibrated case where $\boldsymbol{p} = \hat{\boldsymbol{p}}$ as we study the geometric constraints associated with three views of the same scene. Consider three perspective cameras observing the same point P, whose images are denoted by p_1, p_2 and p_3 (Figure 11.5). The optical centers O_1, O_2 and O_3 of the cameras define a *trifocal plane* T that intersects their retinas along three *trifocal lines* t_1, t_2 and t_3. Each one of these lines passes through the associated epipoles, e.g., the line t_2 associated with the second camera passes through the projections e_{12} and e_{32} of the optical centers of the two other cameras.

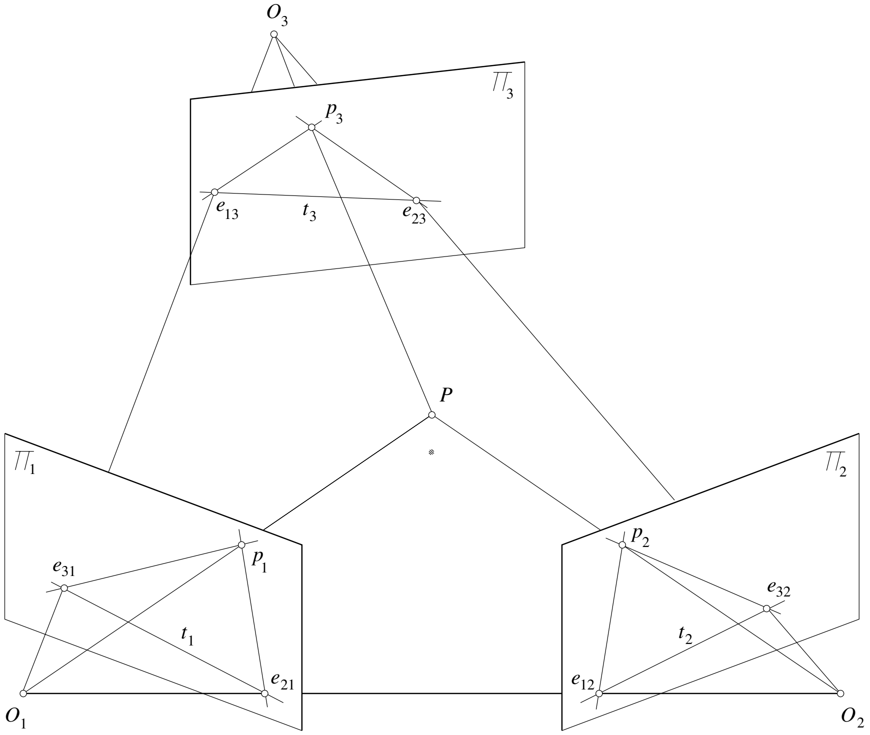


Figure 11.5. Trinocular epipolar geometry.

Each pair of cameras defines an epipolar constraint, i.e.,

$$\begin{cases} \boldsymbol{p}_1^T \mathcal{E}_{12} \boldsymbol{p}_2 = 0, \\ \boldsymbol{p}_2^T \mathcal{E}_{23} \boldsymbol{p}_3 = 0, \\ \boldsymbol{p}_3^T \mathcal{E}_{31} \boldsymbol{p}_1 = 0, \end{cases} \tag{11.2.1}$$

where $\mathcal{E}_{ij}$ denotes the essential matrix associated with the image pairs $i \leftrightarrow j$. These three constraints are not independent since we must have $\boldsymbol{e}_{31}^T \mathcal{E}_{12} \boldsymbol{e}_{32} = \boldsymbol{e}_{12}^T \mathcal{E}_{23} \boldsymbol{e}_{13} = \boldsymbol{e}_{23}^T \mathcal{E}_{31} \boldsymbol{e}_{21} = 0$ (to see why, consider for example the epipoles e_{31} and e_{32}: they are the first and second images of the optical center O_3 of the third camera, and are therefore in epipolar correspondence).

Any two of the equations in (11.2.1) are, on the other hand, independent. In particular, when the essential matrices are known, it is possible to predict the position of the point p_1 from the positions of the two corresponding points p_2 and

p_3: indeed, the first and third constraints in (11.2.1) form a system of two linear equations in the two unknown coordinates of p_1. Geometrically, p_1 is found as the intersection of the epipolar lines associated with p_2 and p_3 (Figure 11.5). Thus the trinocular epipolar geometry offers a solution to the problem of transfer mentioned in the introduction.

11.2.1 Trifocal Geometry

A second set of constraints can be obtained by considering three images of a line instead of a point: as shown in Chapter 4, the set of points that project onto an image line is the plane that contains the line and the pinhole. We can characterize this plane as follows: if $\mathcal{M}$ denotes a 3×4 projection matrix, then a point P projects onto p when $z\boldsymbol{p} = \mathcal{M}\boldsymbol{P}$, or

$$\boldsymbol{l}^T\mathcal{M}\boldsymbol{P} = 0, \qquad (11.2.2)$$

where $\boldsymbol{P} = (x, y, z, 1)^T$ is the 4-vector of homogeneous coordinates of P. Equation (11.2.2) is of course the equation of the plane L that contains both the optical center O of the camera and the line l, and $\boldsymbol{L} = \mathcal{M}^T\boldsymbol{l}$ is the coordinate vector of this plane.

Two images l_1 and l_2 of the same line do not constrain the relative position and orientation of the associated cameras since the corresponding planes L_1 and L_2 always intersect (possibly at infinity). Let us now consider three images l_i, l_2 and l_3 of the same line l and denote by L_1, L_2 and L_3 the associated planes (Figure 11.6). The intersection of these planes forms a line instead of being reduced to a point in the generic case. Algebraically, this means that the system of three equations in three unknowns

$$\begin{pmatrix} \boldsymbol{L}_1^T \\ \boldsymbol{L}_2^T \\ \boldsymbol{L}_3^T \end{pmatrix} \boldsymbol{P} = \boldsymbol{0}$$

must be degenerate, or, equivalently, the rank of the 3×4 matrix

$$\mathcal{L} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{l}_1^T\mathcal{M}_1 \\ \boldsymbol{l}_2^T\mathcal{M}_2 \\ \boldsymbol{l}_3^T\mathcal{M}_3 \end{pmatrix}$$

must be two, which in turn implies that the determinants of all its 3×3 minors must be zero. These determinants are clearly trilinear combinations of the coordinates vectors $\boldsymbol{l}_1$, $\boldsymbol{l}_2$ and $\boldsymbol{l}_3$. As shown below, only two of the four determinants are independent.

11.2.2 The Calibrated Case

To obtain an explicit formula for the trilinear constraints, we pick the coordinate system attached to the first camera as the world reference frame, which allows us to write the projection matrices as

$$\mathcal{M}_1 = (\,\mathrm{Id} \quad \boldsymbol{0}\,), \quad \mathcal{M}_2 = (\,\mathcal{R}_2 \quad \boldsymbol{t}_2\,) \quad \text{and} \quad \mathcal{M}_3 = (\,\mathcal{R}_3 \quad \boldsymbol{t}_3\,),$$

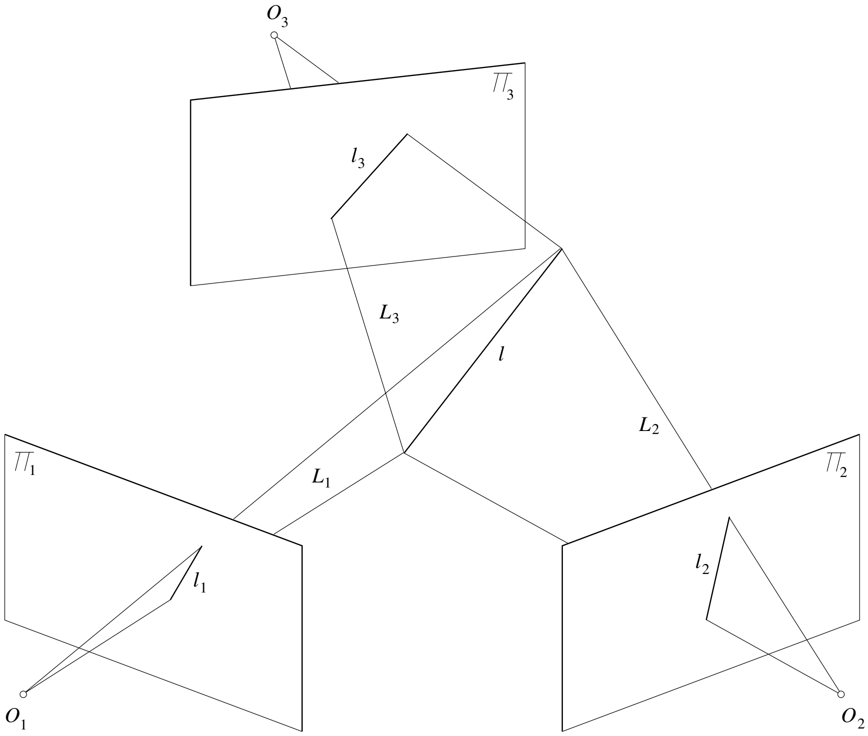


Figure 11.6. Three images of a line define it as the intersection of three planes in the same pencil.

and to rewrite $\mathcal{L}$ as

$$\mathcal{L} = \begin{pmatrix} \boldsymbol{l}_1^T & 0 \\ \boldsymbol{l}_2^T \mathcal{R}_2 & \boldsymbol{l}_2^T \boldsymbol{t}_2 \\ \boldsymbol{l}_3^T \mathcal{R}_3 & \boldsymbol{l}_3^T \boldsymbol{t}_3 \end{pmatrix}. \tag{11.2.3}$$

As shown in the exercises, three of the minor determinants can be written together as

$$\boldsymbol{l}_1 \times \begin{pmatrix} \boldsymbol{l}_2^T \mathcal{G}_1^1 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^2 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^3 \boldsymbol{l}_3 \end{pmatrix} = \boldsymbol{0}, \tag{11.2.4}$$

where

$$\mathcal{G}_1^i = \boldsymbol{t}_2 \boldsymbol{R}_3^{iT} - \boldsymbol{R}_2^i \boldsymbol{t}_3^T \quad \text{for} \quad i = 1, 2, 3, \tag{11.2.5}$$

and $\boldsymbol{R}_2^i$ and $\boldsymbol{R}_3^i$ $(i = 1, 2, 3)$ denote the columns of $\mathcal{R}_2$ and $\mathcal{R}_3$.

The fourth determinant is equal to $|\boldsymbol{l}_1 \; \mathcal{R}_2 \boldsymbol{l}_2 \; \mathcal{R}_3 \boldsymbol{l}_3|$, and it is zero when the normals to the plane L_1, L_2 and L_3 are coplanar. The corresponding equation can be written as a linear combination of the three equations in (11.2.4) (see exercises). Only two of those are linearly independent of course.

Equation (11.2.4) can finally be rewritten as

$$\boldsymbol{l}_1 \approx \begin{pmatrix} \boldsymbol{l}_2^T \mathcal{G}_1^1 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^2 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^3 \boldsymbol{l}_3 \end{pmatrix} \tag{11.2.6}$$

where we use $a \approx b$ to denote that a and b are equal except for an unknown scale factor.

The three 3×3 matrices $\mathcal{G}_1^i$ define the $3\times3\times3$ *trifocal tensor* with 27 coefficients (or 26 up to scale). (A tensor is the multi-dimensional array of coefficients associated with a multilinear form, in the same way that matrices are associated with bilinear forms.)

Since O_1 is the origin of the coordinate system in which all projection equations are expressed, the vectors $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ can be interpreted as the homogeneous image coordinates of the epipoles e_{12} and e_{13}. In particular it follows from (11.2.5) that $\boldsymbol{l}_2^T \mathcal{G}_1^i \boldsymbol{l}_3 = 0$ for any pair of matching epipolar lines l_2 and l_3.

The trifocal tensor also constrains the positions of three corresponding points. Indeed, suppose that P is a point on l. Its first image lies on l_1, so $\boldsymbol{p}_1^T \boldsymbol{l}_1 = 0$ (Figure 11.7). In particular,

$$\boldsymbol{p}_1^T \begin{pmatrix} \boldsymbol{l}_2^T \mathcal{G}_1^1 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^2 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^3 \boldsymbol{l}_3 \end{pmatrix} = 0. \tag{11.2.7}$$

Given three point correspondences $p_1 \leftrightarrow p_2 \leftrightarrow p_3$, we obtain four independent constraints by rewriting (11.2.7) for independent pairs of lines passing through p_2 an p_3, e.g., $\boldsymbol{l}_i' = (1, 0, -u_i)$ and $\boldsymbol{l}_i'' = (0, 1, -v_i)$ $(i = 2, 3)$. These constraints are trilinear in the coordinates of the points p_1, p_2 and p_3.

11.2.3 The Uncalibrated Case

We can still derive trilinear constraints in the image line coordinates when the intrinsic parameters of the three cameras are unknown. Since in this case $\boldsymbol{p} = \mathcal{K}\hat{\boldsymbol{p}}$, and since the image line associated with the vector $\boldsymbol{l}$ is defined by $\boldsymbol{l}^T \boldsymbol{p} = 0$, we obtain immediately $\boldsymbol{l} = \mathcal{K}^{-T}\hat{\boldsymbol{l}}$, or equivalently $\hat{\boldsymbol{l}} = \mathcal{K}^T \boldsymbol{l}$.

In particular, (11.2.3) holds when $\boldsymbol{p}_i = \hat{\boldsymbol{p}}_i$ and $\boldsymbol{l}_i = \hat{\boldsymbol{l}}_i$. In the general case we have

$$\mathcal{L} = \begin{pmatrix} \boldsymbol{l}_1^T \mathcal{K}_1 & 0 \\ \boldsymbol{l}_2^T \mathcal{K}_2 \mathcal{R}_2 & \boldsymbol{l}_2^T \mathcal{K}_2 \boldsymbol{t}_2 \\ \boldsymbol{l}_3^T \mathcal{K}_3 \mathcal{R}_3 & \boldsymbol{l}_3^T \mathcal{K}_3 \boldsymbol{t}_3 \end{pmatrix},$$

and

$$\text{Rank}(\mathcal{L}) = 2 \Longleftrightarrow \text{Rank}(\mathcal{L} \begin{pmatrix} \mathcal{K}_1^{-1} & 0 \\ 0 & 1 \end{pmatrix}) = \text{Rank} \begin{pmatrix} \boldsymbol{l}_1^T & 0 \\ \boldsymbol{l}_2^T \mathcal{A}_2 & \boldsymbol{l}_2^T \boldsymbol{a}_2 \\ \boldsymbol{l}_3^T \mathcal{A}_3 & \boldsymbol{l}_3^T \boldsymbol{a}_3 \end{pmatrix} = 2,$$

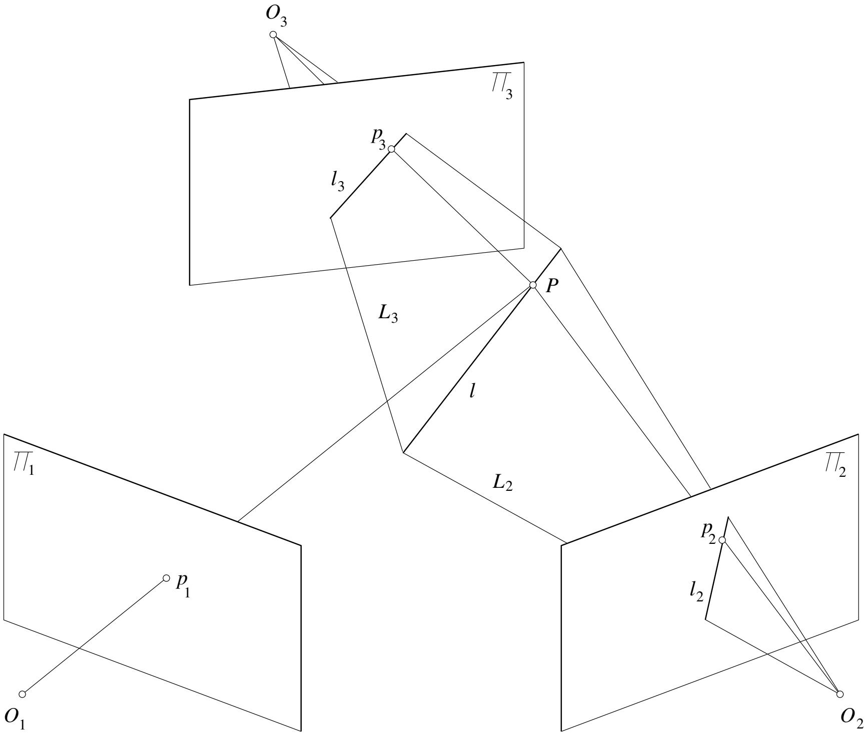


Figure 11.7. Given three images p_1, p_2 and p_3 of the same point P, and two arbitrary images l_2 and l_3 passing through the points p_2 and p_3, the ray passing through O_1 and p_1 must intersect the line where the planes L_2 and L_3 projecting onto l_2 and l_3 meet in space.

where $\mathcal{A}_i \overset{\text{def}}{=} \mathcal{K}_i \mathcal{R}_i \mathcal{K}_1^{-1}$ and $\boldsymbol{a}_i \overset{\text{def}}{=} \mathcal{K}_i \boldsymbol{t}_i$ for $i = 2, 3$. Note that the projection matrices associated with our three cameras are now $\mathcal{M}_1 = (\,\mathcal{K}_1 \quad \mathbf{0}\,)$, $\mathcal{M}_2 = (\,\mathcal{A}_2\mathcal{K}_1 \quad \boldsymbol{a}_2\,)$, and $\mathcal{M}_3 = (\,\mathcal{A}_3\mathcal{K}_1 \quad \boldsymbol{a}_3\,)$. In particular $\boldsymbol{a}_2$ and $\boldsymbol{a}_3$ can still be interpreted as the homogeneous image coordinates of the epipoles e_{12} and e_{13}, and the trilinear constraints (11.2.6) and (11.2.7) still hold when, this time,

$$\mathcal{G}_1^i = \boldsymbol{a}_2 \boldsymbol{A}_3^{iT} - \boldsymbol{A}_2^i \boldsymbol{a}_3^T,$$

where $\boldsymbol{A}_2^i$ and $\boldsymbol{A}_3^i$ $(i = 1, 2, 3)$ denote the columns of $\mathcal{A}_2$ and $\mathcal{A}_3$. As before, we will have $\boldsymbol{l}_2^T \mathcal{G}_1^i \boldsymbol{l}_3 = 0$ for any pair of matching epipolar lines l_2 and l_3.

11.2.4 Estimation of the Trifocal Tensor

We now address the problem of estimating the trifocal tensor from point and line correspondences established across triples of pictures. The equations (11.2.5) defining the tensor are linear in its coefficients and depend only on image measurements.

As in the case of weak calibration, we can therefore use linear methods to estimate these 26 parameters. Each triple of matching points provides four independent linear equations, and every triple of matching lines provides two additional linear constraints. Thus the tensor coefficients can be computed from p points and l lines granted that $2p + l \geq 13$. For example, 7 triples of points or 13 triples of lines will do the trick, as will 3 triples of points and 7 triples of lines, etc.

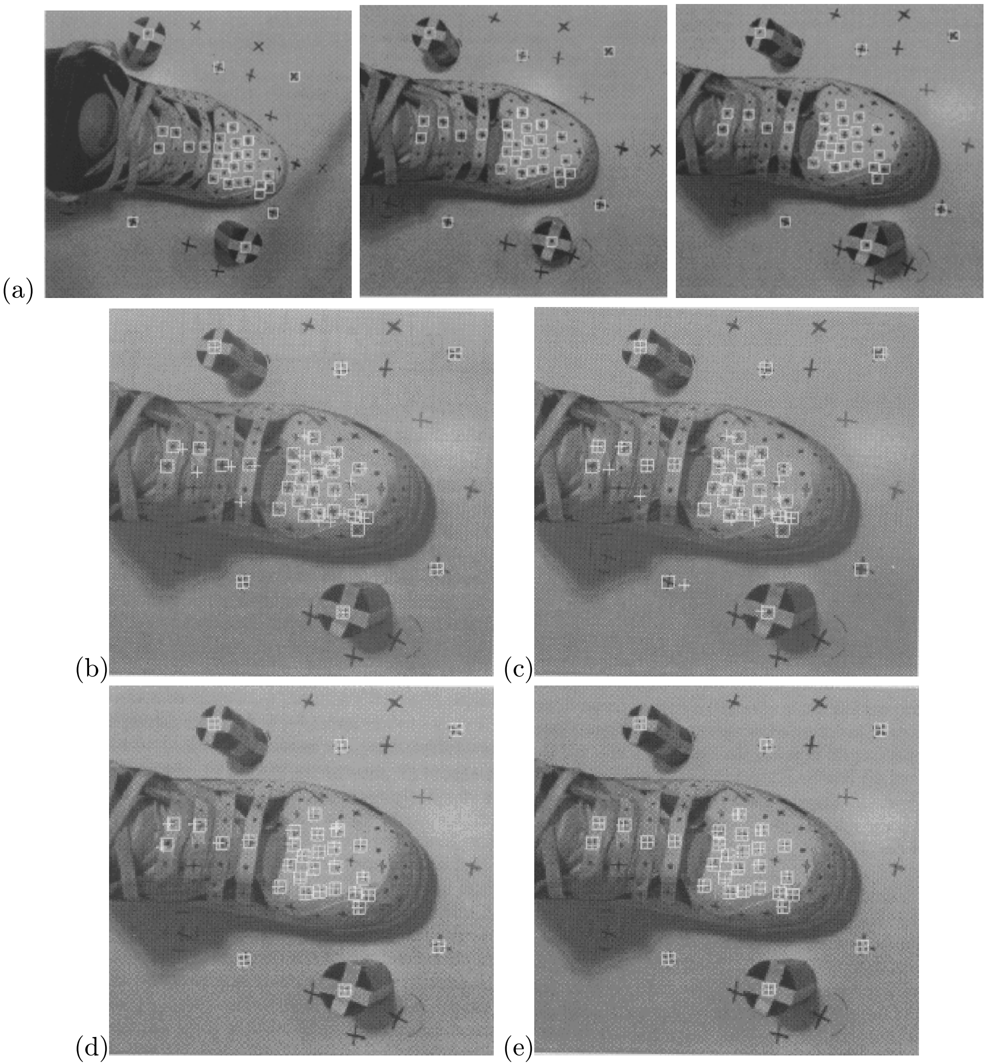
Once the tensor has been estimated, it can be used to predict the position of a point in one image from its positions in the other two. As noted before, the epipolar constraints associated with the camera pairs $1 \leftrightarrow 2$ and $1 \leftrightarrow 3$ can also be used to perform this task. Figure 11.8 shows experimental results using point correspondences found in three images of a sports shoe [?]. It compares the results obtained from the fundamental matrices estimated by the method of Luong *et al.* [?] (Figure 11.8(a)) and by a different weak-calibration technique that takes advantage of the coplanarity of correspondences lying in the ground plane supporting the shoe (see [?] and Figure 11.8(b)) with the results obtained using the trifocal tensor estimated from a minimal set of seven points (Figure 11.8(c)) and a redundant set of ten correspondences (Figure 11.8(d)). In this example, the trifocal tensor clearly gives better results than the fundamental matrices.

As in the case of weak calibration, it is possible to improve the numerical stability of the tensor estimation process by normalizing the image coordinates so the data points are centered at the origin with an average distance from the origin of $\sqrt{2}$ pixel. See [?] for details.

The methods outlined so far ignore the fact that the 26 parameters of the trifocal tensor are *not* independent. This should not come as a surprise: the essential matrix only has five independent coefficients (the associated rotation and translation parameters, the latter being only defined up to scale) and that the fundamental matrix only has seven. Likewise, the parameters defining the trifocal tensor satisfy a number of constraints, including the aforementioned equations $\boldsymbol{l}_2^T \mathcal{G}_1^i \boldsymbol{l}_3 = 0$ $(i = 1, 2, 3)$ satisfied by any pair of matching epipolar lines l_2 and l_3. It is also easy to show that the matrices $\mathcal{G}_1^i$ are singular, a property we will come back to in Chapter 14. Faugeras and Mourrain [?] have shown that the coefficients of the trifocal tensor of an uncalibrated trinocular stereo rig satisfy 8 independent constraints, reducing the total number of independent parameters to 18. The method described in [?] enforces these constraints *a posteriori* by recovering the epipoles e_{12} and e_{13} (or equivalently the vectors $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ in (11.2.5)) from the linearly-estimated trifocal tensor, then recovering in a linear fashion a set of tensor coefficients that satisfy the constraints.

11.3 More Views

What about four views? In this section we follow Faugeras and Mourrain [?] and first note that we can eliminate the depth of the observed point in the projection



Transfer Using Fundamental Matrices	
Method:	Average Distance:
Ground Plane (b)	7.70 pixels
[?] (c)	9.58 pixels

Transfer Using the Trifocal Tensor	
Method:	Average Distance:
7 points (d)	0.98 pixel
10 points (e)	0.44 pixel

Figure 11.8. Transfer experiments: (a) input images; (b)-(c) transfer using the fundamental matrix, estimated in (a) using correspondences on the ground floor and in (b) using the non-linear method of [?]; (d)-(e) transfer using the trifocal tensor estimated in (d) from seven points, and in (e) using least squares from ten points. Reprinted from [?], Figures 2–4. Quantitative comparisons are given in the table, where the average distances between the data points and the transfered ones are shown for each method. The input features are indicated by white squares and the reprojected ones are are indicated by white crosses.

equation by writing

$$z\boldsymbol{p} = \mathcal{M}\boldsymbol{P} \Longleftrightarrow \boldsymbol{p} \times (\mathcal{M}\boldsymbol{P}) = ([\boldsymbol{p}_\times]\mathcal{M})\boldsymbol{P} = 0. \tag{11.3.1}$$

Of course, only two of the scalar equations associated with this vector equation are independent. Choosing (for example) the first and second of these equations allows us to rewrite (11.3.1) as

$$\begin{pmatrix} u\mathcal{M}^3 - \mathcal{M}^1 \\ v\mathcal{M}^3 - \mathcal{M}^2 \end{pmatrix} \boldsymbol{P} = 0, \tag{11.3.2}$$

where $\mathcal{M}^i$ denotes row number i of the matrix $\mathcal{M}$.

Suppose now that we have four views, with associated projection matrices $\mathcal{M}_i$ ($i = 1, 2, 3, 4$). Writing (11.3.2) for each one of these yields

$$\mathcal{Q}\boldsymbol{P} = 0, \quad \text{where} \quad \mathcal{Q} \overset{\text{def}}{=} \begin{pmatrix} u_1\mathcal{M}_1^3 - \mathcal{M}_1^1 \\ v_1\mathcal{M}_1^3 - \mathcal{M}_1^2 \\ u_2\mathcal{M}_2^3 - \mathcal{M}_2^1 \\ v_2\mathcal{M}_2^3 - \mathcal{M}_2^2 \\ u_3\mathcal{M}_3^3 - \mathcal{M}_3^1 \\ v_3\mathcal{M}_3^3 - \mathcal{M}_3^2 \\ u_4\mathcal{M}_4^3 - \mathcal{M}_4^1 \\ v_4\mathcal{M}_4^3 - \mathcal{M}_4^2 \end{pmatrix}. \tag{11.3.3}$$

Equation (11.3.3) is a system of eight homogeneous equations in four unknowns that admits a non-trivial solution. It follows that the rank of the corresponding 8×4 matrix $\mathcal{Q}$ is at most 3, or, equivalently, all its 4×4 minors must have zero determinants. Geometrically, each pair of equations in (11.3.3) represents the ray R_i ($i = 1, 2, 3, 4$) associated with the image point p_i, and $\mathcal{Q}$ must have rank 3 for these rays to intersect at a point P (Figure 11.9).

The matrix $\mathcal{Q}$ has three kinds of 4×4 minors:

1. Those that involve two rows from one projection matrix, and two rows from another one. The equations associated with the six minors of this type include, for example,[5]

$$\operatorname{Det} \begin{pmatrix} u_2\mathcal{M}_1^3 - \mathcal{M}_1^1 \\ v_2\mathcal{M}_1^3 - \mathcal{M}_1^2 \\ u_3\mathcal{M}_2^3 - \mathcal{M}_2^1 \\ v_3\mathcal{M}_2^3 - \mathcal{M}_2^2 \end{pmatrix} = 0. \tag{11.3.4}$$

   These determinants yield bilinear constraints on the position of the associated image points. It is easy to show (see exercises) that the corresponding equations reduce to the epipolar constraints (11.1.2) when we take $\mathcal{M}_1 = (\,\text{Id} \quad 0\,)$ and $\mathcal{M}_2 = (\,\mathcal{R}^T \quad -\mathcal{R}^T\boldsymbol{t}\,)$.

---

[5] General formulas can be given as well by using for example (u^1, u^2) instead of (u, v) and playing around with indices and tensorial notation. We will abstain from this worthy exercise here.

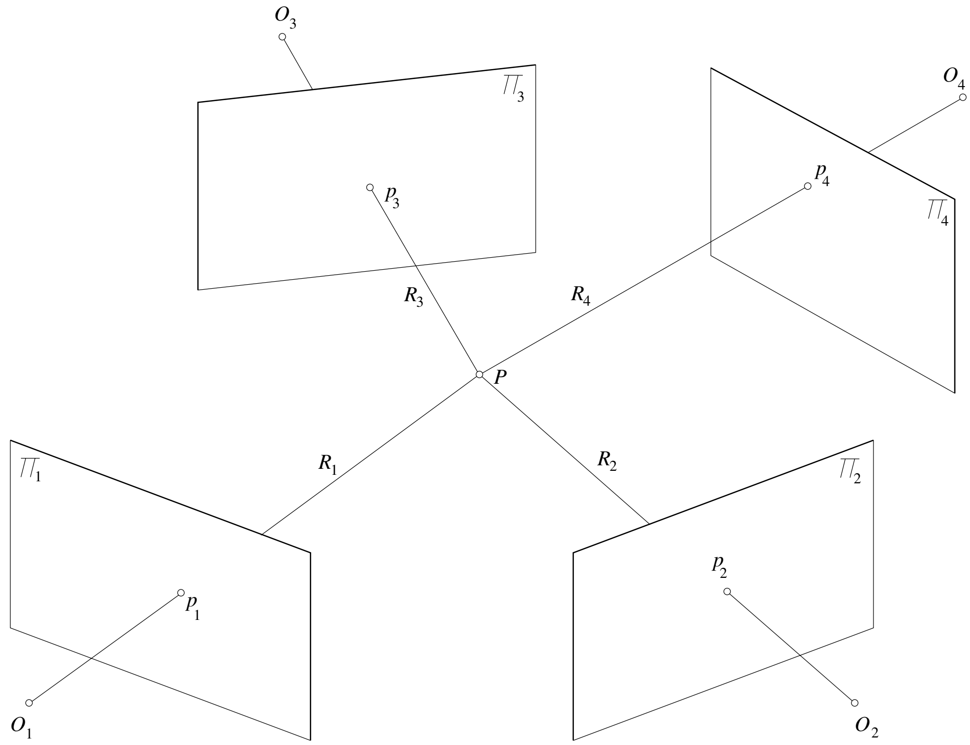


Figure 11.9. Four images p_1, p_2, p_3 and p_4 of the same point P define this point as the intersection of the corresponding rays R_i $(i = 1, 2, 3, 4)$.

2. The second type of minors involves two rows from one projection matrix, and one row from each of two other matrices. There are 48 of those, and the associated equations include, for example,

$$\text{Det}\begin{pmatrix} u_1\mathcal{M}_1^3 - \mathcal{M}_1^1 \\ v_1\mathcal{M}_1^3 - \mathcal{M}_1^2 \\ u_2\mathcal{M}_2^3 - \mathcal{M}_2^1 \\ v_3\mathcal{M}_3^3 - \mathcal{M}_3^2 \end{pmatrix} = 0. \tag{11.3.5}$$

   These minors yield trilinear constraints on the corresponding image positions. It is easy to show (see exercises) that the corresponding equations reduce to the trifocal constraints (11.2.7) introduced in the previous section when we take $\mathcal{M}_1 = (\,\text{Id} \quad 0\,)$. In particular, they can be expressed in terms of the matrices $\mathcal{G}_1^i$ $(i = 1, 2, 3)$. Note that this completes the geometric interpretation of the trifocal constraints, that express here the fact that the rays associated with three images of the same point must intersect in space.

3. The last type of determinant involves one row of each matrix. The equations associated with the 16 minors of this form include, for example,

$$\mathrm{Det}\begin{pmatrix} v_1\mathcal{M}_1^3 - \mathcal{M}_1^2 \\ u_2\mathcal{M}_2^3 - \mathcal{M}_2^1 \\ v_3\mathcal{M}_3^3 - \mathcal{M}_3^2 \\ v_4\mathcal{M}_4^3 - \mathcal{M}_4^2 \end{pmatrix} = 0. \tag{11.3.6}$$

   These equations yield *quadrilinear* constraints on the position of the points $\boldsymbol{p}_i$ $(i = 1, 2, 3, 4)$. Geometrically, each row of the matrix $\mathcal{Q}$ is associated with an image line or equivalently with a plane passing through the optical center of the corresponding camera. Thus each quadrilinearity expresses the fact that the four associated planes intersect in a point (instead of not intersecting at all in the generic case).

Let us focus from now on the the quadrilinear equations. Developing determinants such as (11.3.6) with respect to the image coordinates reveals immediately that the coefficients of the quadrilinear constraints can be written as

$$\varepsilon_{ijkl}\mathrm{Det}\begin{pmatrix} \mathcal{M}_1^i \\ \mathcal{M}_2^j \\ \mathcal{M}_3^k \\ \mathcal{M}_4^l \end{pmatrix}, \tag{11.3.7}$$

where $\varepsilon_{ijkl} = \mp 1$ and i, j, k and l are indices between 1 and 4 (see exercises). These coefficients determine the *quadrifocal tensor* [**?**].

Like its trifocal cousin, this tensor can be interpreted geometrically using both points and lines. In particular, consider four pictures p_i $(i = 1, 2, 3, 4)$ of a point P and four arbitrary image lines l_i passing through these points. The four planes L_i $(i = 1, 2, 3, 4)$ formed by the preimages of the lines must intersect in P, which implies in turn that the 4×4 matrix

$$\mathcal{L} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{l}_1^T\mathcal{M}_1 \\ \boldsymbol{l}_2^T\mathcal{M}_2 \\ \boldsymbol{l}_3^T\mathcal{M}_3 \\ \boldsymbol{l}_4^T\mathcal{M}_4 \end{pmatrix}$$

must have rank 3, and, in particular, that its determinant must be zero. This obviously provides a quadrilinear constraint on the coefficients of the four lines l_i $(i = 1, 2, 3, 4)$. In addition, since each row $\boldsymbol{L}_i^T = \boldsymbol{l}_i^T\mathcal{M}_i$ of $\mathcal{L}$ is a linear combination of the rows of the associated matrix $\mathcal{M}_i$, the coefficients of the quadrilinearities obtained by developing $\mathrm{Det}(\mathcal{L})$ with respect to the coordinates of the lines l_i are simply the coefficients of the quadrifocal tensor as defined by (11.3.7).

Finally, note since $\mathrm{Det}(\mathcal{L})$ is linear in the coordinates of $\boldsymbol{l}_1$, the vanishing of this determinant can be written as $\boldsymbol{l}_1 \cdot \boldsymbol{q}(\boldsymbol{l}_2, \boldsymbol{l}_3, \boldsymbol{l}_4) = 0$, where $\boldsymbol{q}$ is a (trilinear) function of the coordinates of the lines l_i $(i = 2, 3, 4)$. Since this relationship holds for any line

l_1 passing through p_1 it follows that $\boldsymbol{p}_1 \approx \boldsymbol{q}(\boldsymbol{l}_2, \boldsymbol{l}_3, \boldsymbol{l}_4)$. Geometrically, this means that the ray passing through O_1 and p_1 must also pass through the intersection of the planes formed by the preimages of l_2, l_3 and l_4 (Figure 11.10). Algebraically, this means that, given the quadrifocal tensor and arbitrary lines passing through three images of a point, we can predict the position of this point in a fourth image. This provides yet another method for transfer.

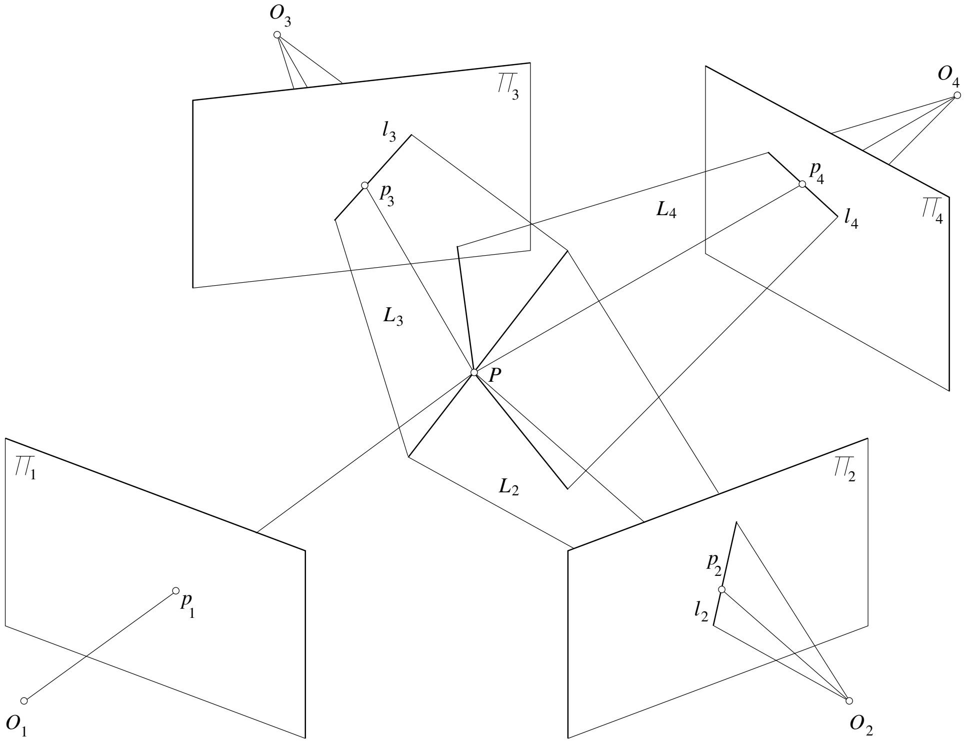


Figure 11.10. Given four images p_1, p_2, p_3 and p_4 of some point P and three arbitrary image lines l_2, l_3 and l_4 passing through the points p_2, p_3 and p_4, the ray passing through O_1 and p_1 must also pass through the point where the three planes L_2, L_3 and L_4 formed by the preimages of these lines intersect.

Note that the quadrifocal constraints are valid in both the calibrated and uncalibrated cases since we have made no assumption on the form of the matrices $\mathcal{M}_i$. The quadrifocal tensor is defined by 81 coefficients (or 80 up to scale), but these coefficients satisfy 51 independent constraints, reducing the total number of independent parameters to 29 [**?**; **?**]. It can also be shown that, although each quadruple of images of the same point yields 16 independent constraints like (11.3.6) on the 80 tensor coefficients, there exists a linear dependency between the 32 equations associated with each pair of points [**?**]. Thus six point correspondences are necessary to estimate the quadrifocal tensor in a linear fashion. Algorithms for performing this task and enforcing the 51 constraints associated with actual quadrifocal tensors can be found in [?].

Finally, Faugeras and Mourrain [?] have shown that the quadrilinear tensor is algebraically dependent on the associated essential/fundamental matrices and trifocal tensor, and thus does not add independent new constraints. Likewise, it can be shown that additional views do not add independent constraints either.

11.4 Notes

The essential matrix as an algebraic form of the epipolar constraint was discovered by Longuet-Higgins [?], and its properties have been elucidated by Huang and Faugeras [?]. The fundamental matrix was introduced by Luong and Faugeras [?; ?]. We will come back to the properties of the fundamental matrix and of the epipolar transformation in Chapter 14, when we adress the problem of recovering the structure of a scene and the motion of a camera from a sequence of perspective images.

The trilinear constraints associated with three views of a line were introduced independently by Spektakis and Aloimonos [?] and Weng, Huang and Ahuja [?] in the context of motion analysis for internally calibrated cameras. They were extended by Shashua [?] and Hartley [?] to the uncalibrated case. The quadrifocal tensor was introduced by Triggs [?]. Geometric studies can by found in Faugeras and Mourrain [?], Faugeras and Papadopoulo [?] and Heyden [?].

We mentioned in the introduction that photogrammetry is concerned with the extraction of quantitative information from multiple pictures. In this context, binocular and trinocular geometric constraints are regarded as the source of *condition equations* that determine the intrinsic and extrinsic parameters (called *interior* and *exterior orientation* parameters in photogrammetry) of a stereo pair or triple. In particular, the Longuet-Higgins relation appears, in a slightly disguised form, as the *coplanarity condition equation*, and trinocular constraints yield *scale-restraint condition equations* that take calibration and image measurement errors into account [?, Chapter X]: in this case, the rays associated with three images of the same point are not guaranteed to intersect anymore (Figure 11.11).

The setup is as follows: if the rays R_1 and R_i ($i = 2, 3$) associated with the image points p_1 and p_i do not intersect, the minimum distance between them is reached at the points P_1 and P_i such that the line joining these points is perpendicular to both R_1 and R_i. Algebraically, this can be written as

$$\overrightarrow{O_1P_1} = z_1^i\overrightarrow{O_1p_1} = \overrightarrow{O_1O_i} + z_i\overrightarrow{O_ip_i} + \lambda_i(\overrightarrow{O_1p_1} \times \overrightarrow{O_ip_i}) \quad \text{for} \quad i = 2, 3. \tag{11.4.1}$$

Assuming that the cameras are internally calibrated so the projection matrices associated with the second and third cameras are $(\mathcal{R}_2^T \quad -\mathcal{R}_2^T\boldsymbol{t}_2)$ and $(\mathcal{R}_3^T \quad -\mathcal{R}_3^T\boldsymbol{t}_3)$, (11.4.1) can be rewritten in the coordinate system attached to the first camera as

$$z_1^i\boldsymbol{p}_1 = \boldsymbol{t}_i + z_i\mathcal{R}_i\boldsymbol{p}_i + \lambda_i(\boldsymbol{p}_1 \times \mathcal{R}_i\boldsymbol{p}_i) \quad \text{for} \quad i = 2, 3. \tag{11.4.2}$$

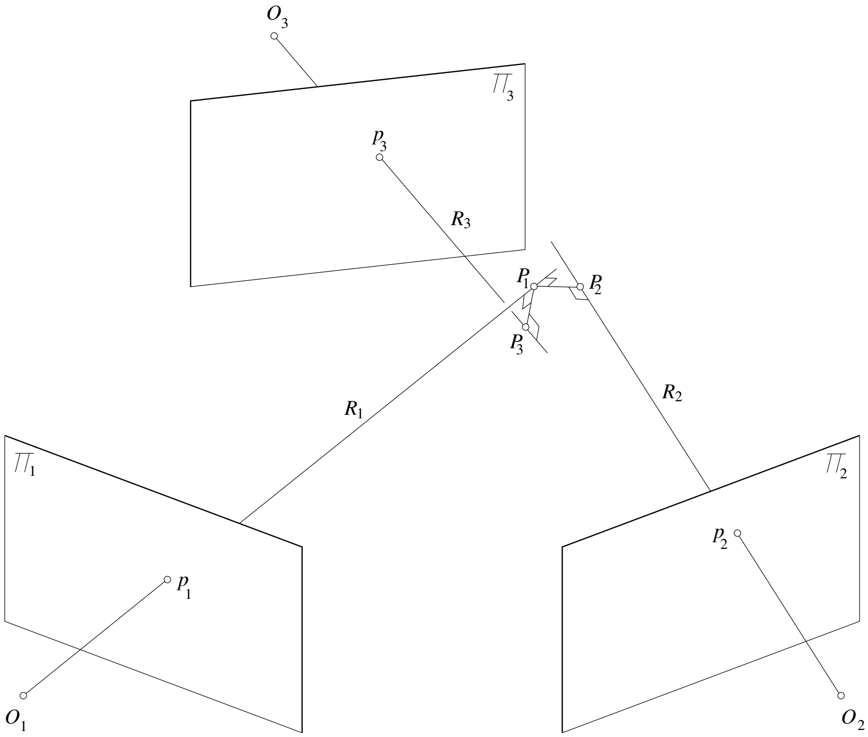


Figure 11.11. Trinocular constraints in the presence of calibration or measurement errors: the rays R_1, R_2 and R_3 may not intersect.

Note that a similar equation could be written as well for completely uncalibrated cameras by including terms depending on the (unknown) intrinsic parameters. In either case, (11.4.2) can be used to calculate the unknowns z_i, λ_i and z_1^i in terms of $\boldsymbol{p}_1$, $\boldsymbol{p}_i$, and the projection matrices associated with the cameras (see exercises). The scale-restraint condition is then written as $z_1^2 = z_1^3$. Although it is more complex than the trifocal constraint (in particular, it is not trilinear in the coordinates of the points p_1, p_2 and p_3), this condition does not involve the coordinates of the observed point, and it can be used (in principle) to estimate the trifocal geometry directly from image data. A potential advantage is that the error function $z_1^2 - z_1^3$ has a clear geometric meaning: it is the difference between the estimates of the depth of P obtained using the pairs of cameras $1 \leftrightarrow 2$ and $1 \leftrightarrow 3$. It would be interesting to further investigate the relationship between the trifocal tensor and the scale-constraint condition, as well as its practical application to the estimation of the trifocal geometry.

11.5 Assignments

Exercises

1. Show that one of the singular values of an essential matrix is 0 and the other two are equal. (Huang and Faugeras [?] have shown that the converse is also true, i.e., any 3 × 3 matrix with one singular value equal to 0 and the other two equal to each other is an essential matrix.)

   Hint: the singular values of $\mathcal{E}$ are the eigenvalues of $\mathcal{E}\mathcal{E}^T$.

   Solution: We have $\mathcal{E} = [\boldsymbol{t}_\times]\mathcal{R}$, thus $\mathcal{E}\mathcal{E}^T = [\boldsymbol{t}_\times][\boldsymbol{t}_\times]^T = [\boldsymbol{t}_\times]^T[\boldsymbol{t}_\times]$. If $\boldsymbol{a}$ is an eigenvector of $\mathcal{E}\mathcal{E}^T$ associated with the eigenvalue λ then, for any vector $\boldsymbol{b}$

   $$\lambda \boldsymbol{b} \cdot \boldsymbol{a} = \boldsymbol{b}^T([\boldsymbol{t}_\times]^T[\boldsymbol{t}_\times]\boldsymbol{a}) = (\boldsymbol{t} \times \boldsymbol{b}) \cdot (\boldsymbol{t} \times \boldsymbol{a}).$$

   Choosing $\boldsymbol{a} = \boldsymbol{b} = \boldsymbol{t}$ shows that $\lambda = 0$ is an eigenvalue of $\mathcal{E}\mathcal{E}^T$. Choosing $\boldsymbol{b} = \boldsymbol{t}$ shows that if $\lambda \neq 0$ then $\boldsymbol{a}$ is orthogonal to $\boldsymbol{t}$. But then choosing $\boldsymbol{a} = \boldsymbol{b}$ shows that

   $$\lambda|\boldsymbol{a}|^2 = |\boldsymbol{t} \times \boldsymbol{a}|^2 = |\boldsymbol{t}|^2|\boldsymbol{a}|^2.$$

   It follows that all non-zero singular values of $\mathcal{E}$ must be equal. Note that the singular values of $\mathcal{E}$ cannot all be zero since this matrix has rank 2.

2. The infinitesimal epipolar constraint (11.1.3) was derived by assuming that the observed scene was static and the camera was moving. Show that when the camera is fixed and the scene is moving with translational velocity $\boldsymbol{v}$ and rotational velocity $\boldsymbol{\omega}$, the epipolar constraint can be rewritten as

   $$\boldsymbol{p}^T([\boldsymbol{v}_\times][\boldsymbol{\omega}_\times])\boldsymbol{p} + (\boldsymbol{p} \times \dot{\boldsymbol{p}}) \cdot \boldsymbol{v} = 0.$$

   Note that this equation is now the sum of the two terms appearing in (11.1.3) instead of their difference.

   Hint: If $\mathcal{R}$ and $\boldsymbol{t}$ denote the rotation matrix and translation vectors appearing in the definition of the essential matrix for a moving camera, show that the object displacement that yields the same motion field for a static camera is given by the rotation matrix $\mathcal{R}^T$ and the translation vector $-\mathcal{R}^T\boldsymbol{t}$.

3. Show that when the 8 × 8 matrix associated with the eight-point algorithm is singular, the eight points and the two optical centers lie on a quadric surface [?].

   Hint: Use the fact that when a matrix is singular, there exists some non-trivial linear combination of its columns that is equal to zero. Also take advantage of the fact that the matrices representing the two projections in the coordinate system of the first camera are in this case $(\,\mathrm{Id} \quad 0\,)$ and $(\,\mathcal{R}^T, -\mathcal{R}^T\boldsymbol{t}\,)$.

4. Show that three of the determinants of the 3 × 3 minors of
$$\mathcal{L} = \begin{pmatrix} \boldsymbol{l}_1^T & 0 \\ \boldsymbol{l}_2^T \mathcal{R}_2 & \boldsymbol{l}_2^T \boldsymbol{t}_2 \\ \boldsymbol{l}_3^T \mathcal{R}_3 & \boldsymbol{l}_3^T \boldsymbol{t}_3 \end{pmatrix}.$$
can be written as
$$\boldsymbol{l}_1 \times \begin{pmatrix} \boldsymbol{l}_2^T \mathcal{G}_1^1 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^2 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^3 \boldsymbol{l}_3 \end{pmatrix} = \mathbf{0}.$$
Show that the fourth determinant can be written as a linear combination of these.

5. Show that (11.3.4) reduces to (11.1.2) when $\mathcal{M}_1 = (\mathrm{Id} \quad 0)$ and $\mathcal{M}_2 = (\mathcal{R}^T \quad -\mathcal{R}^T \boldsymbol{t})$.

6. Show that (11.3.5) reduces to (11.2.7) when $\mathcal{M}_1 = (\mathrm{Id} \quad 0)$.

7. Develop (11.3.6) with respect to the image coordinates and verify that the coefficients can indeed be written in the form (11.3.7).

8. Use (11.4.2) to calculate the unknowns z_i, λ_i and z_1^i in terms of $\boldsymbol{p}_1$, $\boldsymbol{p}_i$, $\mathcal{R}_i$ and $\boldsymbol{t}_i$ $(i = 2, 3)$. Show that the value of λ_i is directly related to the epipolar constraint and characterize the degree of the dependency of $z_1^2 - z_1^3$ on the data points.

Programming Assignments

Note: the assignments below require routines for solving square and overdetermined linear systems. An extensive set of such routines is available in MATLAB as well as in public-domain libraries such as LINPACK and LAPACK that can be downloaded from the Netlib repository (`http://www.netlib.org/`). Data for these assignments will be available in the CD companion to this book.

1. Implement the 8-point algorithm for weak calibration from binocular point correspondences.

2. Implement the linear least-squares version of that algorithm with and without Hartley's pre-conditioning step.

3. Implement an algorithm for estimating the trifocal tensor from point correspondences.

4. Implement an algorithm for estimating the trifocal tensor from line correspondences.

Chapter 12

STEREOPSIS

Fusing the pictures recorded by our two eyes and exploiting the difference (or *disparity*) between them allows us to gain a strong sense of depth (Figure 12.1(left)). This chapter is concerned with the design and implementation of algorithms that mimick our ability to perform this task, known as *stereopsis*. Note that a machine (or for that matter the Martian shown in Figure 12.1(right), or an ordinary spider) may be equipped with three eyes or more, and this will lead us to investigate multi-camera approaches to stereopsis at the end of this chapter.

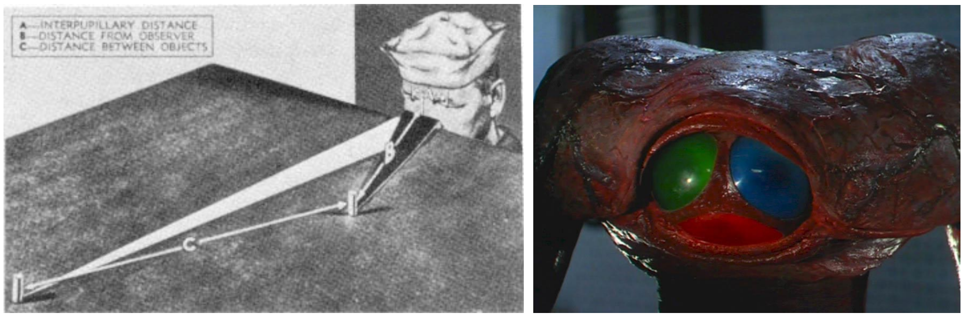


Figure 12.1. The sailor shown in the left picture is, like most people, able to perform stereopsis and gain a sense of depth for the objects within his field of view. Reprinted from [**?**], Figure 6-8. The right photograph is from the 1953 film "The War of the Worlds", and it shows a close-up of the face of a three-eyed Martian warrior. Why such a configuration may prove benefitial will be explained in Section 12.3.1.

Reliable computer programs for stereoscopic perception are of course invaluable in visual robot navigation (Figure 12.2), cartography, aerial reconnaissance and close-range photogrammetry. They are also of great interest in tasks such as image segmentation for object recognition and, as will be seen in Chapter 25, the construction of three-dimensional scene models in image-based rendering, a new discipline that ties together computer vision and computer graphics.

Stereo vision involves two processes: the *binocular fusion* of features observed by the two eyes, and the *reconstruction* of their three-dimensional preimage. The

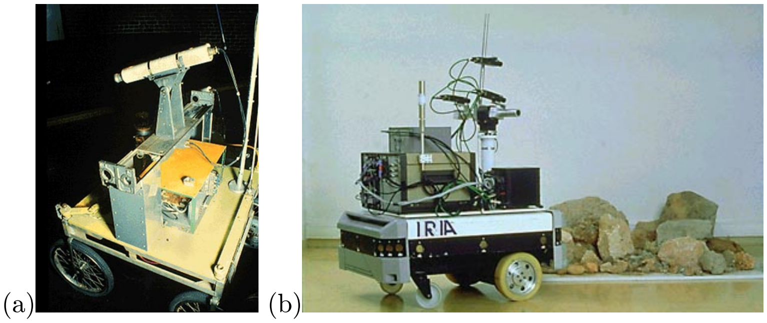


Figure 12.2. Mobile robot navigation is a classical application of stereo vision: (a) the Stanford cart sports a single camera moving in discrete increments along a straight line and providing multiple snapshots of outdoor scenes [**?**]; the INRIA mobile robot uses three cameras to map its environment.

latter is relatively simple: the preimage of matching points can (in principle) be found at the intersection of the rays passing through these points and the associated pupil centers (or pinholes, see Figure 12.3(left)). Thus, when a single image feature is observed at any given time, stereo vision is easy.[1] However, each picture consists of hundreds of thousands of pixels, with tens of thousands of image features such as edge elements, and some method must be devised to establish the correct correspondences and avoid erroneous depth measurements (Figure 12.3(right)).

Although human binocular fusion is effortless and reliabe in most situations, we can be fooled too: the abstract *single-image stereograms* [**?**] that were popular in the late nineties demonstrate this quite well: in this case, repetitive patterns or judiciously assembled random dots are used to trick the eyes into focussing on the wrong correspondences, producing a vivid impression of layered planes.[2] This suggests that constructing a reliable stereo vision program is difficult, a fact that will be attested time and again in the rest of this chapter. As should be expected, the geometric machinery introduced in Chapter 11 will prove extremely useful in tackling this problem. We will assume in the rest of this chapter that all cameras have been carefully calibrated so their intrinsic and extrinsic parameters are precisely known relative to some fixed world coordinate system. The case of multiple uncalibrated cameras will be examined in the context of structure from motion in Chapters 13 and 14.

---

[1] This is actually how some laser range finders work: two cameras observe an object while a laser beam scans its surface one point at a time. After thresholding the two pictures, the bright laser spot is, effectively, the only surface point seen by the cameras. See Chapter 23 for details.

[2] To enjoy this effect without any special equipment or expensive props, you may try to sit down in a place decorated with a repetitive tile pattern such as those often found in bathroom floors. By letting your mind wander and your eyes unfocus, you may be able to see the floor jump up by a foot or so, and even pass your hand through the "virtual" floor. This experiment, best conducted late at night, is quite worth the effort.

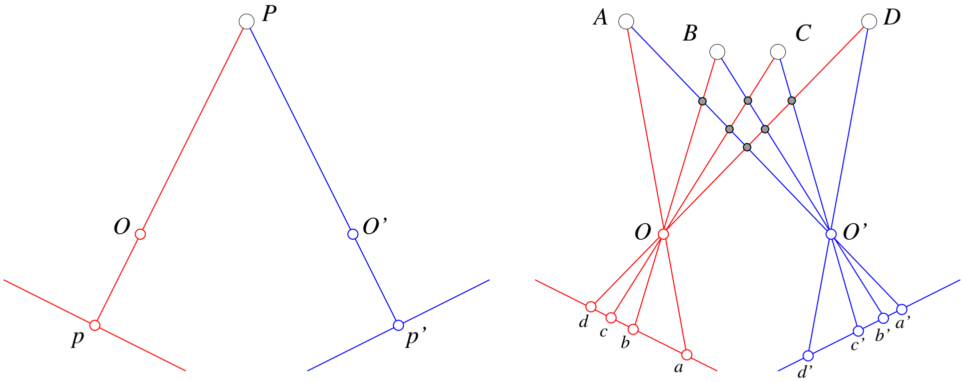


Figure 12.3. The binocular fusion problem: in the simple case of the diagram shown on the left, there is no ambiguity and stereo reconstruction is a simple matter. In the more usual case shown on the right, any of the four points in the left picture may, a priori, match any of the four points in the right one. Only four of these correspondences are correct, the other ones yielding the incorrect reconstructions shown as small grey discs.

12.1 Reconstruction

Given a calibrated stereo rig and two matching image points p and p', it is in principle straightforward to reconstruct the corresponding scene point by intersecting the two rays $R = Op$ and $R' = O'p'$. However, the rays R and R' will never, in practice, actually intersect, due to calibration and feature localization errors (Figure 12.4). In this context, various reasonable approaches to the reconstruction problem can be adopted. For example, we may choose to construct the line segment perpendicular to R and R' that intersects both rays: the mid-point P of this segment is the closest point to the two rays and can be taken as the pre-image of p and p'. It should be noted that a similar construction was used at the end of Chapter 11 to characterize algebraically the geometry of multiple views in the presence of calibration or measurement errors. The equations (11.4.1) and (11.4.2) derived in that chapter are readily adapted to the calculation of the coordinates of P in the frame attached to the first camera.

Alternatively, we can reconstruct a scene point using a purely algebraic approach: given the projection matrices $\mathcal{M}$ and $\mathcal{M}'$ and the matching points p and p', we can rewrite the constraints $z\boldsymbol{p} = \mathcal{M}\boldsymbol{P}$ and $z'\boldsymbol{p}' = \mathcal{M}\boldsymbol{P}$ as

$$\left\{ \begin{array}{l} \boldsymbol{p} \times \mathcal{M}\boldsymbol{P} = 0 \\ \boldsymbol{p}' \times \mathcal{M}'\boldsymbol{P} = 0 \end{array} \right. \iff \begin{pmatrix} [\boldsymbol{p}_\times]\mathcal{M} \\ [\boldsymbol{p}'_\times]\mathcal{M}' \end{pmatrix} \boldsymbol{P} = 0.$$

This is an overconstrained system of four independent linear equations in the homogeneous coordinates of P, that is easily solved using the linear least-squares techniques introduced in Chapter 5. Unlike the previous approach, this reconstruction method does not have an obvious geometric interpretation, but it generalizes

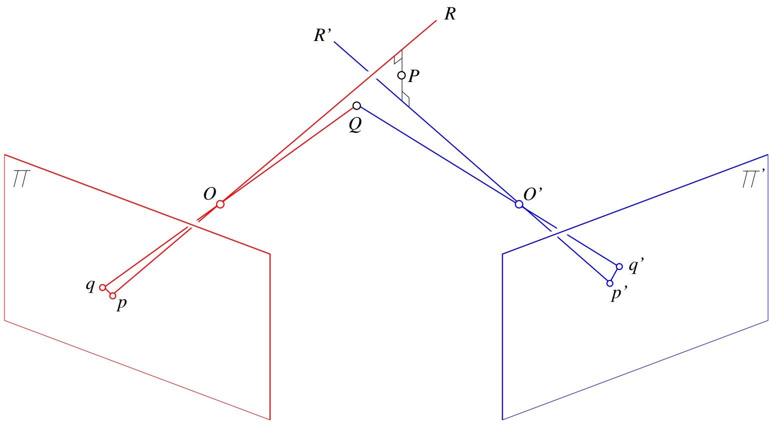


Figure 12.4. Triangulation in the presence of measurement errors. See text for details.

readily to the case of three or more cameras, each new picture simply adding two additional constraints.

Finally, we can reconstruct the scene point associated with p and p' as the point Q with images q and q' that minimizes $d^2(p,q) + d^2(p',q')$ (Figure 12.4). Unlike the two other methods presented in this section, this approach does not allow the closed-form computation of the reconstructed point, which must be estimated via non-linear least-squares techniques such as those introduced in Chapter 5. The reconstruction obtained by either of the other two methods can be used as a reasonable guess to initialize the optimization process. This non-linear approach also readily generalizes to the case of multiple images.

Before moving on to studying the problem of binocular fusion, let us now say a few words about two key components of stereo vision systems: camera calibration and image rectification.

12.1.1 Camera Calibration

As noted in the introduction, we will assume throughout this chapter that all cameras have been carefully calibrated (using, for example, one of the techniques introduced in Chapter 5) so their intrinsic and extrinsic parameters are precisely known relative to some fixed world coordinate system. This is of course a prerequisite for the reconstruction methods presented in the previous section since they require that the projection matrices associated with the two cameras be known, or, equivalently, that a definite ray be associated with every image point. It should also be noted that, once the intrinsic and extrinsic camera parameters are known, it is a simple matter to estimate the multi-view geometry (essential matrix for two views, trifocal tensor for three, etc.) as described in Chapter 11. This will play a fundamental role in the algorithms for establishing stereo correspondences presented in Sections 12.2 and 12.3.

12.1.2 Image Rectification

The calculations associated with stereo algorithms are often considerably simplified when the images of interest have been rectified, i.e., replaced by two projectively equivalent pictures with a common image plane parallel to the baseline joining the two optical centers (Figure 12.5). The rectification process can be implemented by projecting the original pictures onto the new image plane. With an apropriate choice of coordinate system, the rectified epipolar lines are scanlines of the new images, and they are also parallel to the baseline.

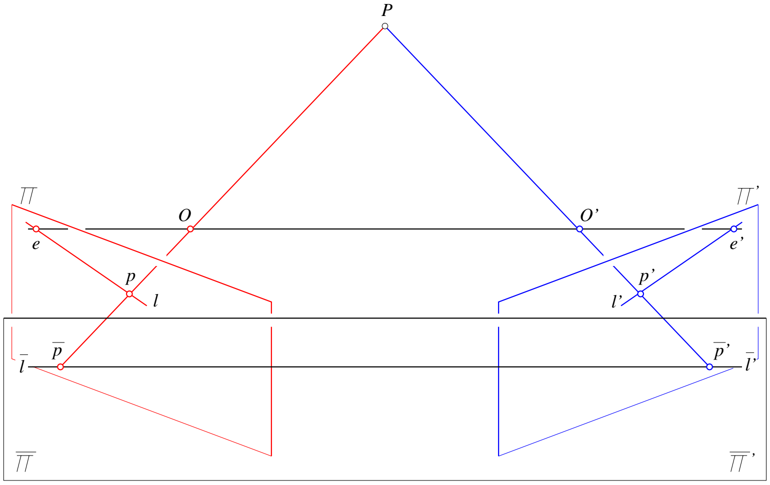


Figure 12.5. A rectified stereo pair: the two image planes Π and Π' are reprojected onto a common plane $\bar{\Pi} = \bar{\Pi}'$ parallel to the baseline. The epipolar lines l and l' associated with the points p and p' in the two pictures map onto a common scanline $\bar{l} = \bar{l}'$ also parallel to the baseline and passing through the reprojected points $\bar{p}$ and $\bar{p}'$. The rectified images are easily constructed by considering each input image as a polyhedral mesh and using texture mapping to render the projection of this mesh into the plane $\bar{\Pi} = \bar{\Pi}'$.

As noted in [**?**], there are two degrees of freedom involved in the choice of the rectified image plane: (1) the distance between this plane and the baseline, which is essentially irrelevant since modifying it will only change the scale of the rectified pictures, an effect easily balanced by an inverse scaling of the image coordinate axes, and (2) the direction of the rectified plane normal in the plane perpendicular to the baseline. Natural choices include picking a plane parallel to the line where the two original retinas intersect, and minimizing the distortion associated with the reprojection process.

In the case of rectified images, the notion of disparity introduced informally earlier takes a precise meaning: given two points p and p' located on the same scanline of the left and right images, with coordinates (u, v) and (u', v), the disparity is defined as the difference $d = u' - u$. Let us assume from now on normalized image coordinates. If B denotes the distance between the optical centers, also called baseline in this context, it is easy to show that the depth of P in the (normalized) coordinate system attached to the first camera is $z = -B/d$ (Figure 12.6).

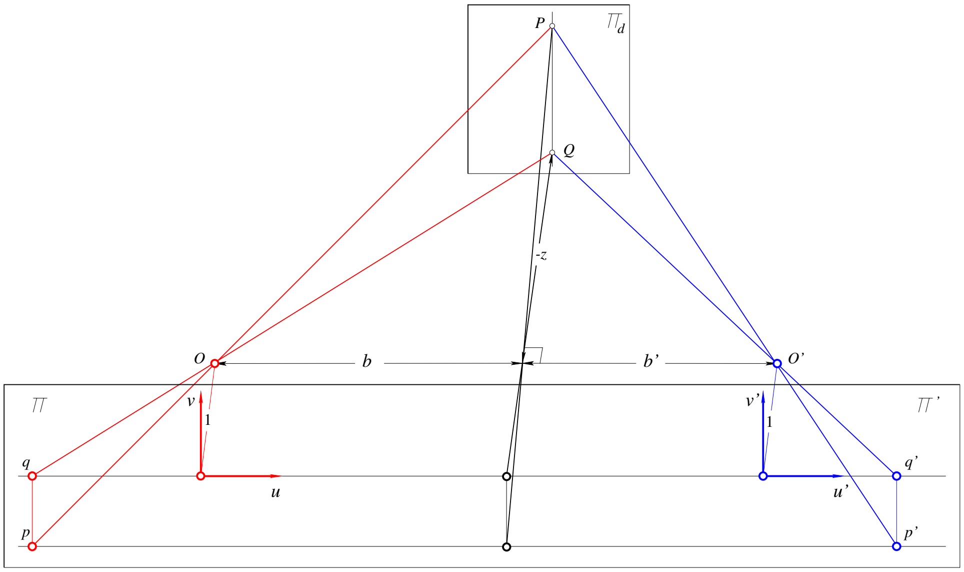


Figure 12.6. Triangulation for rectified images: the rays associated with two points p and p' on the same scanline are by construction guaranteed to intersect in some point P. As shown in the text, the depth of P relative to the coordinate system attached to the left camera is inversely proportional to the disparity $d = u' - u$. In particular, the preimage of all pairs of image points with constant disparity d is a *frontoparallel* plane Π_d (i.e., a plane parallel to the camera retinas).

To show this, let us consider first the points q and q' with coordinates $(u, 0)$ and $(u', 0)$, and the corresponding scene point Q. Let b and b' denote the respective distances between the orthogonal projection of Q onto the baseline and the two optical centers O and O'. The triangles qQq' and OQO' are similar, and it follows immediately that $b = zu$ and $b' = -zu'$. Thus $B = -zd$, which proves the result for q and q'. The general case involving p and p' with $v \neq 0$ follows immediately from the fact that the line PQ is parallel to the two lines pq and $p'q'$ and therefore also parallel to the rectified image plane. In particuliar, the coordinate vector of the point P in the frame attached to the first camera is $\boldsymbol{P} = -(B/d)\boldsymbol{p}$, where $\boldsymbol{p} = (u, v, 1)^T$ is the vector of normalized image coordinates of p. This provides yet another reconstruction method for rectified stereo pairs.

Human Vision: Stereopsis

Before moving on to algorithms for establishing binocular correspondences, let us pause for a moment to discuss the mechanisms underlying human stereopsis. First, it should be noted that, unlike the cameras rigidly attached to a passive stereo rig, the two eyes of a person can rotate in their sockets. At each instant, they *fixate* on a particular point in space, i.e., they rotate so that its two images form in the centers of the eyes' foveas. Figure 12.7 illustrates a simplified, two-dimensional situation.

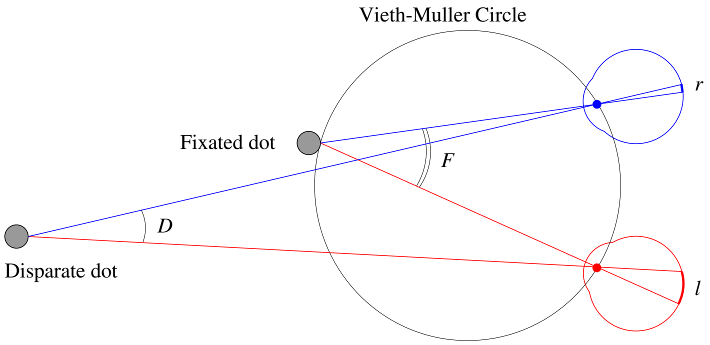


Figure 12.7. This diagram depicts a situation similar to that of the sailor in Figure 12.1. The close-by dot is fixated by the eyes, and it projects onto the center of their foveas, with no disparity. The two images of the far dot deviate from this central position by different amounts, indicating a different depth.

If l and r denote the (counterclockwise) angles between the vertical planes of symmetry of two eyes and two rays passing through the same scene point, we define the corresponding disparity as $d = r - l$ (Figure 12.7). It is an elementary exercise in trigonometry to show that $d = D - F$, where D denotes the angle between these rays, and F is the angle between the two rays passing through the fixated point. Points with zero disparity lie on the *Vieth-Müller circle* that passes through the fixated point and the anterior nodal points of the eyes. Points lying inside this circle have a positive (or *convergent*) disparity, points lying outside it have, as in Figure 12.7, a negative (or *divergent*) disparity,[3] and the locus of all points having a given disparity d forms, as d varies, the pencil of all circles passing through the two eyes' nodal points. This property is clearly sufficient to rank-order in depth dots that are near the fixation point. However, it is also clear that the *vergence angles* between the vertical *median plane* of symmetry of the head and the two fixation rays must be known in order to reconstruct the absolute position of scene points.

The three-dimensional case is naturally a bit more complicated, the locus of zero-disparity points becoming a surface, the *horopter*, but the general conclusion is the same, and absolute positioning requires the vergence angles. As already demonstrated by Wundt and Helmholtz [**?**, pp. 313-314] a hundred years ago, there is strong evidence that these angles cannot be measured very accurately by our nervous system. In fact, the human visual system can be fooled into believing that threads that actually lie in the same vertical plane lie instead on a convex or concave surface, depending on the distance between the observer and this plane [**?**, pp. 318-321]. Likewise, the *relief* models used in sculpture to mimick solids with much reduced depths are almost indistinguishable binocularly from the originals (see [**?**, pp. 324-326] for an analytical justification). On the other hand, *relative* depth, or rank-ordering of points along the line of sight, can be judged quite accurately: for example, it is possible to decide which one of two targets near the horopter is closer to an observer for disparities of a few seconds of arc (*stereoacuity threshold*), which matches the minimum separation that can be measured with one eye (*monocular hyperacuity threshold*) [**?**, p. 307] (though the stereo disparity threshold increases quickly as one gets away from the horopter, see, for example, [**?**]). It can therefore reasonably be argued that the output of human stereopsis consists mostly of a map of *relative* depth information, conveying a partial depth order between scene points [**?**, pp. 176-177].[4] In that context, the main role of eye movements in stereopsis would be to bring the images within *Panum's fusional area*, a disc with a diameter of 6min of arc in the fovea center where fusion can occur [**?**, pp. 148] (points can still be vividly perceived in depth for much larger disparities, but they will appear as double images, a phenomenon known as *diplopia*).

---

[3] The terminology comes from the fact that the eyes would have to converge (resp. diverge) to fixate on a point inside (resp. outside) the Vieth-Müller circle. Note that the position of this circle in space depends on the fixation point (even if the fixation angle F is preserved), since the rotation centers of the eyes do not coincide with their anterior nodal points.

Concerning the construction of correspondences between the left and right images, Julesz [**?**] asks the following question: is the basic mechanism for binocular fusion a monocular process (where local brightness patterns (micropatterns) or higher organizations of points into objects (macropatterns) are identified *before* being fused), a binocular one (where the two images are combined into a single field where all further processing takes place), or a combination of both? Some anecdotal evidence hints at a binocular mechanism, for example, to quote Julesz [**?**, pp. 1133-1134]: "In aerial reconnaissance it is known that objects camouflaged by a complex background are very difficult to detect but jump out if viewed sterescopically." But this is not conclusive: "Though the macropattern (hidden object) is *difficult* to see monocularly, it *can* be seen. Therefore, the evidence is not sufficient to prove that depth can be perceived without monocular macropattern recognition." To gather more conclusive data, Julesz [**?**] introduces a new device, the *random dot stereogram*, a pair of synthetic images obtained by randomly spraying black dots on white objects, typically a small square plate floating over a larger one (Figure 12.8).

To quote Julesz [**?**, p. 1127-1128] again: "When viewed monocularly, the images appear completely random. But when viewed stereoscopically, the image pair gives the impression of a square markedly in front of (or behind) the surround. ... Of course, depth perception under these conditions takes longer to establish because of the absence of monocular cues. Still, once depth is perceived, it is quite stable. This experiment shows quite clearly that it is possible to perceive depth without monocular macropatterns." By locally perturbing the stereograms in various ways, Julesz proceeds to show that the identification of monocular micropatterns is not necessary for depth perception either. Although monocular perception is certainly also involved in most situations (e.g., making the central region in each image visible by increasing its average brightness has the effect of speeding up depth perception), the conclusion, articulated in [**?**], is clear: human binocular fusion cannot be explained by peripheral processes directly associated with the physical retinas. Intead, it must involve the central nervous system and an imaginary *cyclopean retina* that combines the left and

---

[4] Frisby [**?**, p. 155] goes even further, suggesting that the depth effect might be a secondary advantage of stereopsis, the primary one being to give the human visual system an effective way of performing grouping and segmentation.

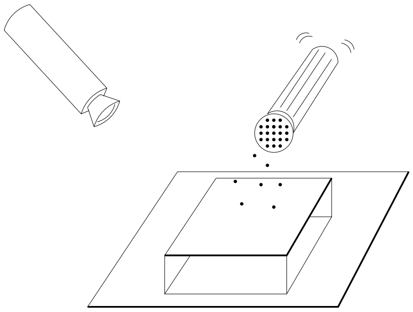


Figure 12.8. Creating random dot stereograms by shaking pepper over a pair of plates observed by two cameras. In the experiments presented in [**?**], the two images are of course synthesized by a computer using a random-number generator to decide the dot locations and pixel intensities, that can either be binary values as in the situation described in the text, or more generally random values in the 0..15 range. The two pictures have the same random background and differ in a central region by a constant horizontal offset.

right image stimuli as a single unit.

Julesz has proposed two models of human stereopsis. The first one represents the binocular field in terms of a finite number of *difference fields* formed by substracting from the first picture the second one shifted by various degrees of disparity [**?**]. The matching process amounts in this case to finding various patterns in some of the difference fields. This model has been implemented in the AUTOMAP-1 program that has proven capable of fusing simple randon dot stereograms [**?**]. The second model represents each image by a rectangular array of compass needles (or *dipoles*) mounted on spherical joints. A black dot will force the corresponding dipole to point north, and a white dot will force it to point south. After the directions of all dipoles are set, they are coupled to their four neighbors via springs. Finally, the two dipole arrays are superimposed, and left to follow each other's magnetic attraction under various horizontal shifts.

These two models are *cooperative*, with neighboring matches influencing each other to avoid ambiguities and promote a global analysis of the observed scene. The approach proposed by Marr and Poggio [**?**] is another instance of such a cooperative process. Their algorithm relies on three constraints: (1) *compatibility* (black dots can only match black dots, or more generally, two image features can only match if they have possibly arisen from the same physical marking), (2) *uniqueness* (a black dot in one image matches at most one black dot in the other picture), and (3) *continuity* (the disparity of matches varies smoothly almost everywhere in the image). Given a number of black dots on a pair of corresponding epipolar lines, Marr and Poggio build a graph that reflects possible correspondences (Figure 12.9).

The nodes of the graph are pairs of black dots within some disparity range, reflecting the compatibility constraint; vertical and horizontal arcs represent inhibitory connections associated with the uniqueness constraint (any match between two dots should discourage any other match for both the left dot –horizontal inhibition– and the right one –vertical inhibition– in the pair); and diagonal arcs represent excitory connections associated with the continuity constraint (any match should favor nearby matches with similar disparities).

In this approach, a quality measure is associated with each node. It is initialized to 1

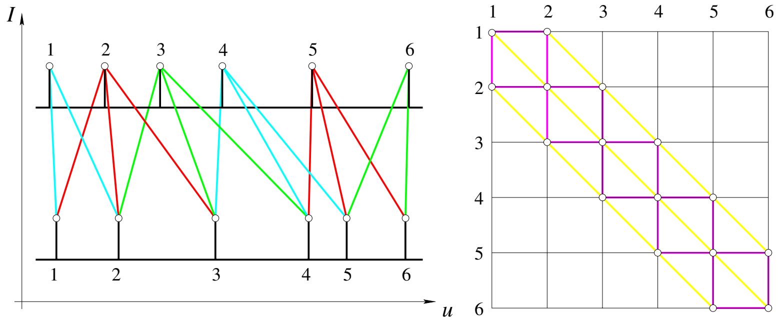


Figure 12.9. A cooperative approach to stereopsis: the Marr-Poggio algorithm [?]. The left part of the figure shows two intensity profiles along the same scanline of two images. The spikes correspond to black dots. The line segments joining the two profiles indicate possible matches between dots given some maximum disparity range. These matches are also shown in the right part of the figure, where they form the nodes of a graph. The vertical and horizontal arcs of this graph join nodes associated with the same dot in the left or right image. The diagonal arcs join nodes with similar disparities.

for every pair of potential matches within some disparity range. The matching process is iterative and parallel, each node being assigned at each iteration a weighted combination of its neighbors' values. Excitory connections are assigned weights equal to 1, and inhibitory ones weights equal to 0. A node is assigned a value of 1 when the corresponding weighted sum exceeds some threshold, and a value of 0 otherwise. This approach works quite reliably on random dot stereograms (Figure 12.10), but not on natural images, perhaps, as suggested by Faugeras [?], because the constraints it enforces are not sufficient to deal with the complexities of real pictures. Section 12.2 will present a number of algorithms that perform better on most real images, but the original Marr-Poggio algorithm and its implementation retain the interest of offering an early example of a theory of human stereopsis that allows the fusion of random dot stereograms.

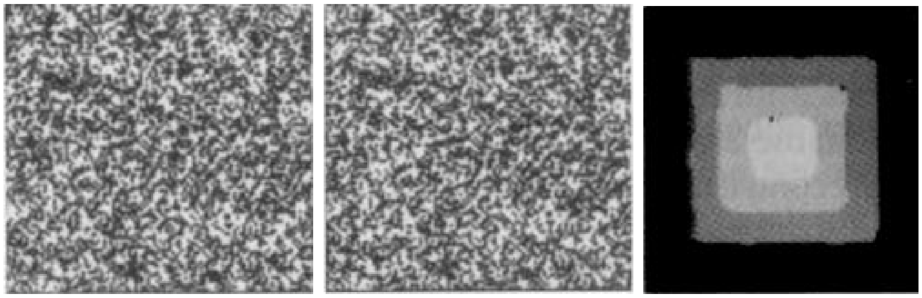


Figure 12.10. From left to right: a random dot stereogram depicting four planes at varying depth (a "wedding cake") and the disparity map obtained after 14 iterations of the Marr-Poggio cooperative algorithm. Reprinted from [?], Figure 3-7.

12.2 Binocular Fusion

12.2.1 Correlation

Correlation methods find pixel-wise image correspondences by comparing intensity profiles in the neighborhood of potential matches, and they are amongst the first techniques ever proposed to solve the binocular fusion problem [**?**; **?**]. More precisely, let us consider a *rectified* stereo pair and a point (u, v) in the first image. We associate with the window of size $p = (2m+1) \times (2n+1)$ centered in (u, v) the vector $\boldsymbol{w}(u, v) \in \mathbb{R}^p$ obtained by scanning the window values one row at a time (the order is in fact irrelevant as long as it is fixed). Now, given a potential match $(u+d, v)$ in the second image, we can construct a second vector $\boldsymbol{w}'(u+d, v)$ and define the corresponding (normalized) correlation function as

$$C(d) = \frac{1}{|\boldsymbol{w} - \bar{\boldsymbol{w}}|} \frac{1}{|\boldsymbol{w}' - \bar{\boldsymbol{w}}'|} (\boldsymbol{w} - \bar{\boldsymbol{w}}) \cdot (\boldsymbol{w}' - \bar{\boldsymbol{w}}'),$$

where the u, v and d indices have been omitted for the sake of conciseness and $\bar{\boldsymbol{a}}$ denotes the vector whose coordinates are all equal to the mean of the coordinates of $\boldsymbol{a}$ (Figure 12.11).

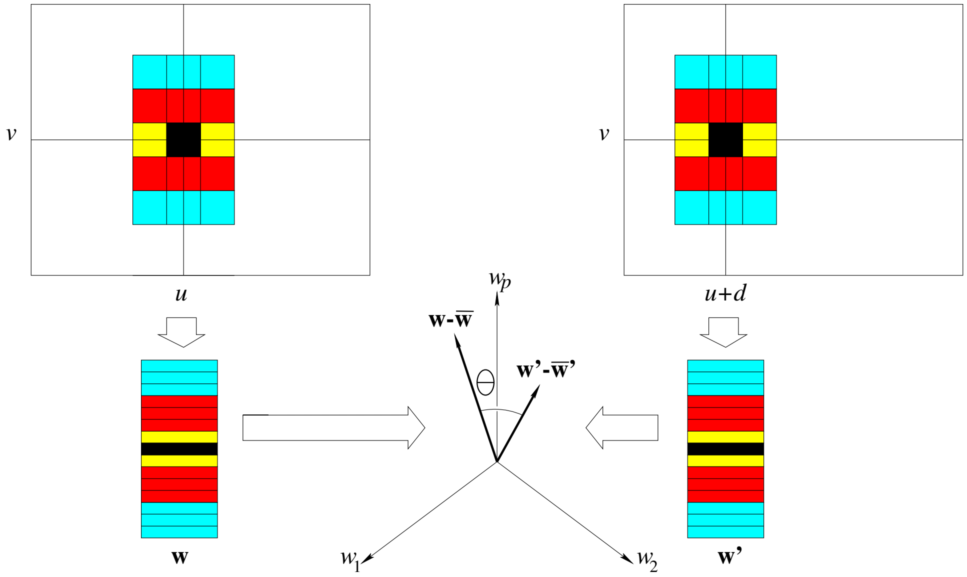


Figure 12.11. Correlation of two 3×5 windows along corresponding epipolar lines. The second window position is separated from the first one by an offset d. The two windows are encoded by vectors $\boldsymbol{w}$ and $\boldsymbol{w}'$ in $\mathbb{R}^{15}$, and the correlation function measures the cosine of the angle θ between the vectors $\boldsymbol{w} - \bar{\boldsymbol{w}}$ and $\boldsymbol{w}' - \bar{\boldsymbol{w}}'$ obtained by substracting from the components of $\boldsymbol{w}$ and $\boldsymbol{w}'$ the average intensity in the corresponding windows.

The normalized correlation function C clearly ranges from -1 to $+1$, and it reaches its maximum value when the image brightnesses of the two windows are related by an affine transformation $I' = \lambda I + \mu$ for some constants λ and μ with $\lambda > 0$ (see exercises). In other words, maxima of this function correspond to image patches separated by a constant offset and a positive scale factor, and stereo matches can be found by seeking the maximum of the C function over some pre-determined range of disparities.[5]

At this point, let us make a few remarks about matching methods based on correlation. First, it is easily shown (see exercises) that maximizing the correlation function is equivalent to minimizing the norm of the difference between the vectors $(1/|\boldsymbol{w}-\bar{\boldsymbol{w}}|)(\boldsymbol{w}-\bar{\boldsymbol{w}})$ and $(1/|\boldsymbol{w}'-\bar{\boldsymbol{w}}'|)(\boldsymbol{w}'-\bar{\boldsymbol{w}}')$, or equivalently the sum of the squared differences between the pixel values of the normalized windows being compared. Second, although the calculation of the normalized correlation function at every pixel of an image for some range of disparities is computationally expensive, it can be implemented efficiently using recursive techniques (see exercises). Finally, a major problem with correlation-based techniques for establishig stereo correspondences is that they implicitly assume that the observed surface is (locally) parallel to the two image planes (Figure 12.12). This suggests a two-pass algorithm where initial estimates of the disparity are used to warp the correlation windows to compensate for inequal amounts of foreshortening in the two pictures [**?**; **?**].

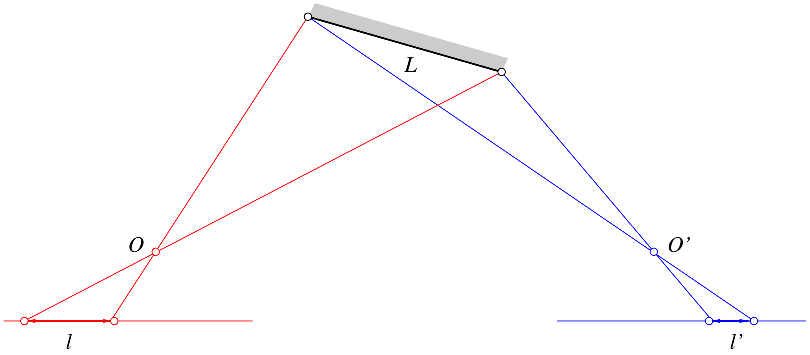


Figure 12.12. The foreshortening of non-frontoparallel surfaces is different for the two cameras: a surface segment with length L projects onto two image segments of different lengths l and l'.

Figure 12.13 shows a reconstruction example obtained by such a method [**?**]. In this case, a warped window is associated in the right image with each rectangle in the left image. This window is defined by the disparity in the center of the rectangle and its derivatives. An optimization process is used to find the values of the disparity and of its derivatives that maximize the correlation between the left rectangle and the right window, using interpolation to retrieve appropriate values in the right image (see exercises for more details). As shown in Figure 12.13, the reconstruction obtained by this method is clearly better than the reconstruction found by plain correlation.

---

[5] The invariance of C to affine transformations of the brightness function affords correlation-based matching techniques some degree of robustness in situations where the observed surface is not quite Lambertian, or the two cameras have different gains or lenses with different f stops.

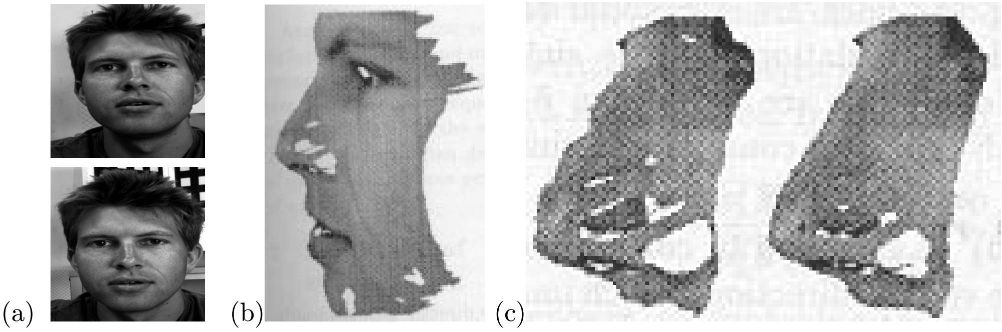


Figure 12.13. Correlation-based stereo matching: (a) a pair of stereo pictures; (b) a texture-mapped view of the reconstructed surface; (c) comparison of the regular (left) and refined (right) correlation methods in the nose region. Reprinted from [**?**], Figures 5, 8 and 9.

12.2.2 Multi-Scale Edge Matching

We saw in the last section that slanted surfaces pose problems to correlation-based matchers. Other arguments against correlation can be found in the works of Julesz [**?**, p. 1145] ("One might think that the matching of corresponding point domains (instead of corresponding patterns)[6] could be achieved by searching for a best fit according to some similarity criterion (e.g., maximal cross-correlation). ... But such a process cannot work. If the zone [used to search for correspondences] is small, noise can easily destroy any zone-matching; if the zone size is increased, ambiguities arise at the boundaries of objects which are at different distances.") and Marr [**?**, p. 105] ("...by and large the primitives that the processes operate on should correspond to physical items that have identifiable physical properties and occupy a definite location on a surface in the world. Thus one should not try to carry out stereo matching between gray-level intensity arrays, precisely because a pixel corresponds only implicitly and not explicitly to a location on a visible surface."). These arguments suggest that correspondences should be found at a variety of scales, and that matches between (hopefully) physically-significant image features such as edges should be prefered to matches between raw pixel intensities. Marr and Poggio [**?**] propose an algorithm that follows these two principles. Its overall structure is quite simple, as described below.

---

[6] This remark shows, by the way, that the random dot stereogram experiments of Julesz do not dismiss, at least in his thought, the possibility of a correlation-based process as opposed to a higher-level, pattern recognition one.

1. Convolve the two (rectified) images with $\nabla^2 G_\sigma$ filters of increasing standard deviations $\sigma_1 < \sigma_2 < \sigma_3 < \sigma_4$.
2. Find zero crossings of the Laplacian along horizontal scanlines of the filtered images.
3. For each filter scale σ, match zero crossings with the same parity and roughly equal orientations in a $[-w_\sigma, +w_\sigma]$ disparity range, with $w_\sigma = 2\sqrt{2}\sigma$.
4. Use the disparities found at larger scales to control eye vergence and cause unmatched regions at smaller scales to come into correspondence.

Algorithm 12.1: *The Marr-Poggio-Grimson multi-scale algorithm for establishing stereo correspondences [**?**; **?**].*

Note that matches are sought at each scale in the $[-w_\sigma, w_\sigma]$ disparity range, where $w_\sigma = 2\sqrt{2}\sigma$ is the width of the central negative portion of the $\nabla^2 G_\sigma$ filter. This choice is motivated by psychophysical and statistical considerations. In particular, assuming that the convolved images are white Gaussian processes, Grimson [**?**] has shown that the probability of a false match occurring in the $[-w_\sigma, +w_\sigma]$ disparity range of a given zero crossing is only 0.2 when the orientations of the matched features are within 30° of each other. A simple mechanism can be used to disambiguate the multiple potential matches that may still occur within the matching range. See [**?**] for details.

Of course, limiting the search for matches to the $[-w_\sigma, +w_\sigma]$ range prevents the algorithm from matching *correct* pairs of zero crossings whose disparity falls outside this interval. Since w_σ is proportional to the scale σ at which matches are sought, eye movements (or equivalently image offsets) controlled by the disparities found at large scales must be used to bring large-disparity pairs of zero crossings within matchable range at a fine scale. This process occurs in Step 4 of the algorithm, and it is illustrated by Figure 12.14. Once matches have been found, the corresponding disparities can be stored in a buffer, called the $2\frac{1}{2}$-*dimensional sketch* by Marr and Nishihara [**?**].

This algorithm has been implemented by Grimson [**?**], and extensively tested on random dot stereograms and natural images. An example appears in Figure 12.15.

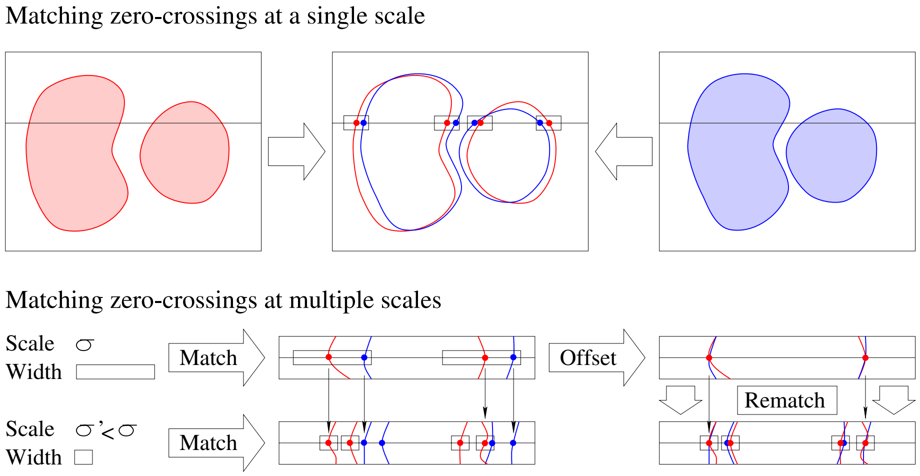

Matching zero-crossings at a single scale
Matching zero-crossings at multiple scales
Scale σ
Width
Match
Offset
Rematch
Scale σ'<σ
Width
Match

Figure 12.14. Multi-scale matching of zero crossings: the eye movements (or equivalently the image offsets used in matching) are controlled by seeking image regions that have been assigned a disparity value at a scale σ' but not at a scale $\sigma < \sigma'$. These values are used to refine the eye positions and bring the corresponding regions within matchable range. The disparity value associated with a region can be found by various methods, for example by averaging the disparity values found at each matched zero crossing within it.

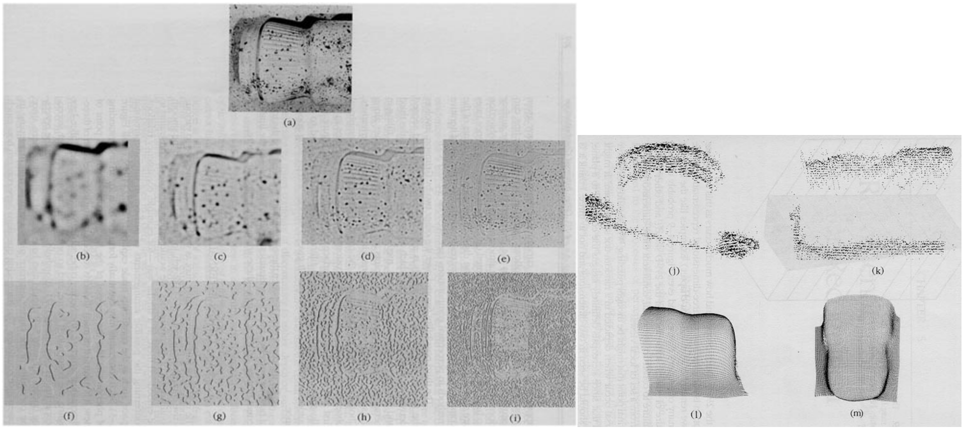

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)

Figure 12.15. Applying the multi-scale matching algorithm of Marr and Poggio [**?**] to a pair of images: (a) one of the pictures in the stereo pair; (b)-(e) its convolution with four ∇^2_σ filters of increasing sizes; (f)-(i) the corresponding zero crossings; (j)-(k) two views of the disparity map obtained after matching; (l)-(m) two views of the surface obtained by interpolating the reconstructed dots using the algorithm described in [**?**]. Reprinted from [**?**], Figure 4-8.

12.2.3 Dynamic Programming

It is reasonable to assume that the order of matching image features along a pair of epipolar lines is the inverse of the order of the corresponding surface attributes along the curve where the epipolar plane intersects the observed object's boundary (Figure 12.16(left)). This is the so-called *ordering constraint* that has been used in stereo circles since the early eighties [**?**; **?**]. Interestingly enough, this constraint may not be satisfied by real scenes, in particular when small solids occlude parts of larger ones (Figure 12.16(right)), or more rarely, at least in robot vision, when transparent objects are involved.

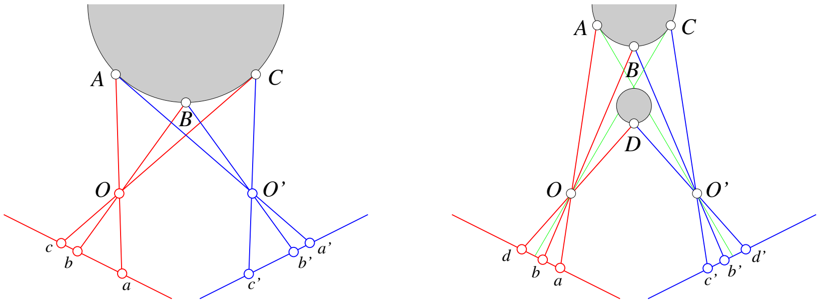


Figure 12.16. Ordering constraints. In the (usual) case shown in the left part of the diagram, the order of feature points along the two (oriented) epipolar lines is the same, and it is the inverse of the order of the scene points along the curve where the observed surface intersects the epipolar plane. In the case shown in the right part of the figure, a small object lies in front of a larger one. Some of the surface points are not visible in one of the images (e.g., A is not visible in the right image), and the order of the image points is not the same in the two pictures: b is on the right of d in the left image, but b' is on the left of d' in the right image.

Despite these reservations, the ordering constraint remains a reasonable one, and it can be used to devise efficient algorithms relying on *dynamic programming* [**?**; **?**] to establish stereo correspondences (Figure 12.17). Specifically, let us assume that a number of feature points (say edgels) have been found on corresponding epipolar lines. Our objective here is to match the intervals separating those points along the two intensity profiles (Figure 12.17(left)). According to the ordering constraint, the order of the feature points must be the same, although the occasional interval in either image may be reduced to a single point corresponding to missing correspondences associated with occlusion and/or noise.

This setting allows us to restate the matching problem as the optimization of a path's cost over a graph whose nodes correspond to pairs of left and right image features, and arcs represent matches between left and right intensity profile intervals bounded by the features of the corresponding nodes (Figure 12.17(right)). This optimization problem can be solved using dynamic programming as shown in Algorithm 12.2 below.

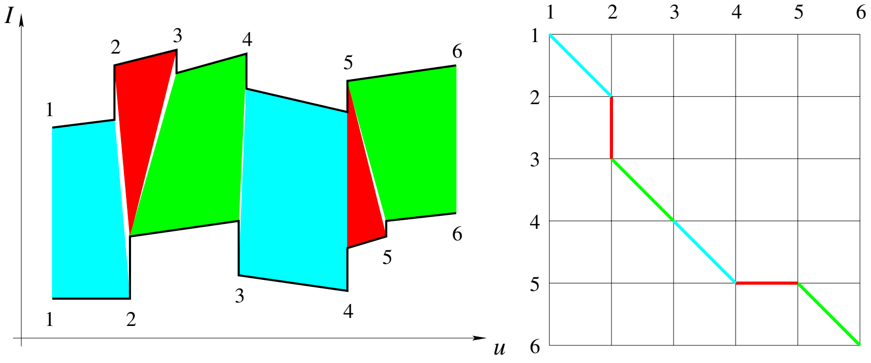


Figure 12.17. Dynamic programming and stereopsis: the left part of the figure shows two intensity profiles along matching epipolar lines. The polygons joining the two profiles indicate matches between successive intervals (some of the matched intervals may have zero length). The right part of the diagram represents the same information in graphical form: an arc (thick line segment) joins two nodes (i, i') and (j, j') when the intervals (i, j) and (i', j') of the intensity profiles match each other.

```
% Loop over all nodes $(k, l)$ in ascending order.
for $k = 1$ to $m$ do
   for $l = 1$ to $n$ do
      % Initialize optimal cost $C(k, l)$ and backward pointer $B(k, l)$.
      $C(k, l) \leftarrow +\infty; B(k, l) \leftarrow$ nil;
      % Loop over all inferior neighbors $(i, j)$ of $(k, l)$.
      for $(i, j) \in$ Inferior-Neighbors$(k, l)$ do
         % Compute new path cost and update backward pointer if necessary.
         $d \leftarrow C(i, j) +$ Arc-Cost$(i, j, k, l)$;
         if $d < C(k, l)$ then $C(k, l) \leftarrow d; B(k, l) \leftarrow (i, j)$ endif;
         endfor;
      endfor;
   endfor;
% Construct optimal path by following backward pointers from $(m, n)$.
$P \leftarrow \{(m, n)\}; (i, j) \leftarrow (m, n)$;
while $B(i, j) \neq$ nil do $(i, j) \leftarrow B(i, j); P \leftarrow \{(i, j)\} \cup P$ endwhile.
```

Algorithm 12.2: *A dynamic-programming algorithm for establishing stereo correspondences between two corresponding scanlines with* m *and* n *edge points respectively (the endpoints of the scanlines are included for convenience). Two auxiliary functions are used:* Inferior-Neighbors(k, l) *returns the list of neighbors* (i, j) *of the node* (k, l) *such that* $i \leq k$ *and* $j \leq l$*, and* Arc-Cost(i, j, k, l) *evaluates and returns the cost of matching the intervals* (i, k) *and* (j, l)*. For correctness,* $C(1, 1)$ *should be initialized with a value of zero.*

As given, Algorithm 12.2 has a computational complexity of $O(mn)$, where m and n respectively denote the number of edge points on the matched left and right scanlines.[7] Variants of this approach have been implemented by Baker and Binford [?], who combine a coarse-to-fine intra-scanline search procedure with a cooperative process for enforcing inter-scanline consistency, and Ohta and Kanade [?], who use dynamic programming for both intra- and inter-scanline optimization, the latter procedure being conducted in a three-dimensional search space. Figure 12.18 shows a sample result taken from [?].

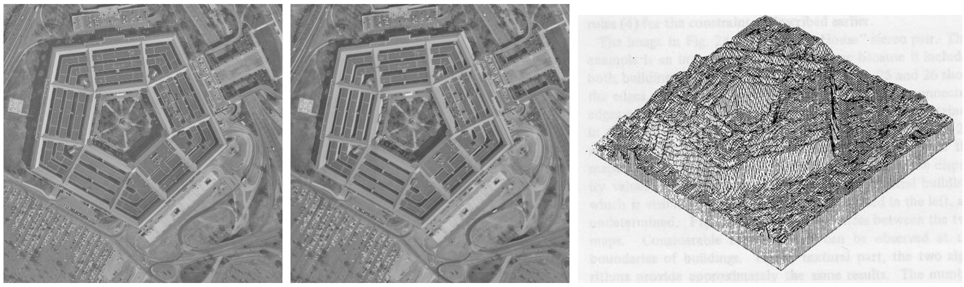


Figure 12.18. Two images of the Pentagon and an isometric plot of the disparity map computed by the dynamic-programming algorithm of Ohta and Kanade [?]. Reprinted from [?], Figures 18 and 22.

12.3 Using More Cameras

12.3.1 Trinocular Stereo

Adding a third camera eliminates (in large part) the ambiguity inherent in two-view point matching. In essence, the third image can be used to check hypothetical matches between the first two pictures (Figure 12.19): the three-dimensional point associated with such a match is first reconstructed then reprojected into the third image. If no compatible point lies nearby, then the match must be wrong. In fact, the reconstruction/reprojection process can be avoided by noting, as in Chapter 11, that, given three weakly (and a fortiori strongly) calibrated cameras and two images of a point, one can always predict its position in a third image by intersecting the corresponding epipolar lines.

The trifocal tensor introduced in Chapter 11 can be used to also predict the tangent line to some image curve in one image given the corresponding tangents in the other images (Figure 12.20): given matching tangents l_2 and l_3 in images 2 and 3, we can reconstruct the tangent l_1 in image number 1 using Eq. (11.2.4),

---

[7] Our version of the algorithm assumes that all edges are matched. To account for noise and edge detection errors, it is reasonable to allow the matching algorithm to skip a bounded number of edges, but this does not change its asymptotic complexity [?].

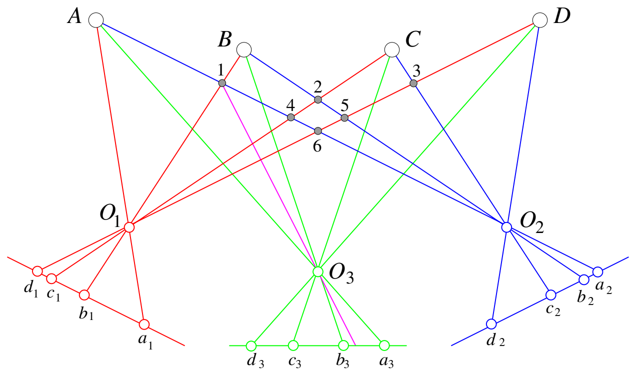


Figure 12.19. The small grey discs indicate the incorrect reconstructions associated with the left and right images of four points. The addition of a central camera removes the matching ambiguity: none of the corresponding rays intersects any of the six discs. Alternatively, matches between points in the first two images can be checked by reprojecting the corresponding three-dimensional point in the third image. For example, the match between b_1 and a_2 is obviously wrong since there is no feature point in the third image near the reprojection of the hypothetical reconstruction numbered 1 in the diagram.

rewritten here as:

$$\boldsymbol{l}_1 \approx \begin{pmatrix} \boldsymbol{l}_2^T \mathcal{G}_1^1 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^2 \boldsymbol{l}_3 \\ \boldsymbol{l}_2^T \mathcal{G}_1^3 \boldsymbol{l}_3 \end{pmatrix}, \quad \text{where} \quad \mathcal{G}_1^i = \boldsymbol{t}_2 \boldsymbol{R}_3^{iT} - \boldsymbol{R}_2^i \boldsymbol{t}_3^T \quad \text{for} \quad i = 1, 2, 3,$$

$\boldsymbol{R}_2^i$ and $\boldsymbol{R}_3^i$ $(i = 1, 2, 3)$ denote the columns of the rotation matrices $\mathcal{R}_2$ and $\mathcal{R}_3$ associated with cameras 2 and 3, and $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ denote the corresponding translation vectors (here "$\approx$" is used to denote equality up to scale).

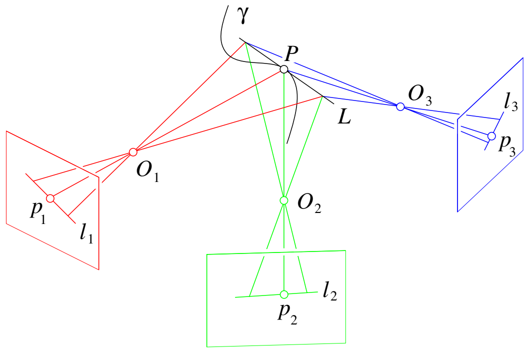


Figure 12.20. Given matches between the points p_1 and p_2 and their tangents l_1 and l_2 in two images, it is possible to predict both the position of the corresponding point p_3 and tangent l_3 in a third image.

Algorithms for trinocular stereo include [?; ?; ?; ?]. An example is shown in Figure 12.21.

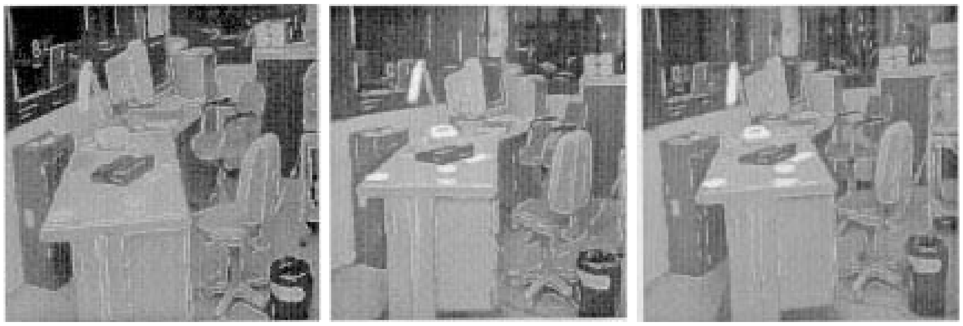


Figure 12.21. Three images and the correspondences between edges found by the algorithm of Robert and Faugeras [?; ?]. Reprinted from [?], Figure 9.

As shown in [?; ?], it is in fact also possible to predict the curvature at a point on some image curve given the corresponding curvatures in the other images (see exercises). This fact can be used to effectively reconstruct curves from their images [?; ?].

12.3.2 Multiple-Baseline Stereo

In most trinocular stereo algorithms, potential correspondences are hypothesized using two of the images, then confirmed or rejected using the third one. In contrast, Okutami and Kanade [?] have proposed a a multi-camera method where matches are found using all pictures at the same time. The basic idea is simple but elegant: assuming that all the images have been rectified, the search for the correct disparities is replaced by a search for the correct depth, or rather its inverse. Of course, the inverse depth is proportional to the disparity for each camera, but the disparity varies from camera to camera, and the inverse depth can be used as a common search index. Picking the first image as a reference, Okutami and Kanade add the sums of squared differences associated with all other cameras into a global evaluation function E (this is of course, as shown earlier, equivalent to adding the correlation functions associated with the images).

Figure 12.22 plots the value of E as a function of inverse depth for various sets of cameras. It should be noted that the corresponding images contain a repetitive pattern and that using only two or three cameras does not yield a single, well-defined minimum. On the other hand, adding more cameras provides a clear minimum corresponding to the correct match.

Figure 12.23 shows a sequence of ten rectified images and a plot of the surface reconstructed by the algorithm.

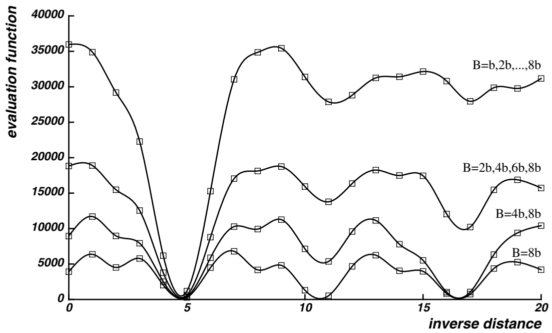


Figure 12.22. Combining multiple-baseline stereo pairs: the sum of squared differences is plotted here as a function of the inverse depth for various numbers of input pictures. The data are taken from a scanline near the top of the images shown in Figure 12.23, whose intensity is nearly periodic. The diagram clearly shows that the mininum of the function becomes less and less ambiguous as more images are added. Reprinted from [?], Figure 7.

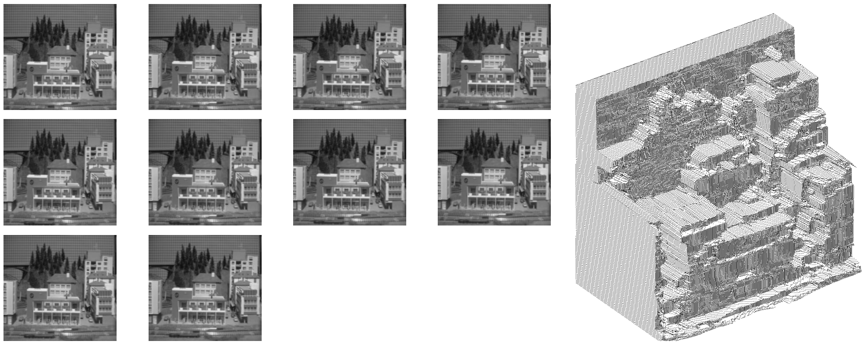


Figure 12.23. A series of ten images and the corresponding reconstruction. The gridboard near the top of the images is the source for the nearly periodic brightness signal giving rise to ambiguities in Figure 12.22. Reprinted from [?], Figure 13(c).

12.4 Notes

The fact that disparity gives rise to stereopsis in human beings was first demonstrated by Wheatstone's invention of the stereoscope [?]. The fact that disparity is sufficient for stereopsis without eye movements was demonstrated shortly afterwards by Dove [?], using illumination provided by an electric spark and much too brief for eye vergence to take place [?, p. 455]. Human stereo vision is further discussed in the classical works of Helmholtz [?] and Julesz [?] as well as the books by Frisby [?] and Marr [?]. Theories of human binocular perception not presented in this chapter for lack of space include [?; ?; ?].

Excellent treatments of machine stereopsis can be found in the books of Grimson [**?**], Marr [**?**], Horn [**?**] and Faugeras [**?**]. Marr focusses on the computational aspects of human stereo vision, while Horn's account emphasizes the role of photogrammetry in artificial stereo systems. Grimson and Faugeras emphasize the geometric and algorithmic aspects of stereopsis. The constraints associated with stereo matching are discussed in [**?**].

As noted earlier, image edges are often used as the basis for establishing binocular correspondences, at least in part because they can (in principle) be identified with physical properties of the imaging process, corresponding for example to albedo, color, or occlusion boundaries. A point rarely taken into account by stereo matching algorithms is that binocular fusion *always* fails along the contours of solids bounded by smooth surfaces (Figure 12.24). Indeed, the corresponding image edges are in this case viewpoint dependent, and matching them yields erroneous reconstructions.

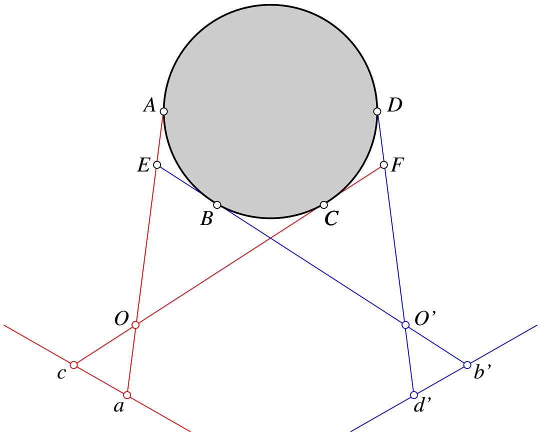


Figure 12.24. Stereo matching fails at smooth object boundaries: for narrow baselines, the pairs (c, d') and (a, b') will be easily matched by most edge-based algorithms, yielding the fictitious points F and E as the corresponding three-dimensional reconstructions.

As shown in [**?**; **?**; **?**; **?**] and the exercises, three cameras are sufficient in this case to reconstruct a local second-degree surface model.

It is not quite clear at this point whether feature-based matching is preferable to grey-level matching. The former is accurate near surface markings but only yields a sparse set of measurements, while the latter may give poor results in uniform regions but provides dense correspondences in textured areas. In this context, the topic of dense surface interpolation from sparse samples is important, although it has hardly been mentioned in this chapter. The interested reader is refered to [**?**; **?**] for more details.

A different approach to stereo vision that we have also failed to discuss for lack of space involves higher-level interpretation processes, for example prediction/verification methods operating on graphical image descriptions [**?**], or hierarchical techniques matching curves, surfaces and volumes found in two images [**?**].

All of the algorithms presented in this chapter (implicitly) assume that the images being fused are quite similar. This is equivalent to considering a short baseline. An effective algorithm for dealing with wide baselines can be found in [**?**]. Another, model-based approach will be discussed in Chapter 25.

Finally, we have limited our attention to stereo rigs with fixed intrinsic and extrinsic parameters. *Active vision* is concerned with the construction of vision systems capable of dynamically modifying these parameters, e.g., changing camera zoom and vergence angles, and taking advantage of these capabilities in perceptual and robotic tasks [**?**; **?**; **?**; **?**].

12.5 Assignments

Exercises

1. Use the definition of disparity to characterize the accuracy of stereo reconstruction as a function of baseline and depth.

2. Give reconstruction formulas for verging eyes in the plane.

3. Give an algorithm for generating an ambiguous random dot stereogram that can depict two different planes hovering over a third one.

4. Give an algorithm for generating single-image random dot stereograms.

5. Show that the correlation function reaches its maximum value of 1 when the image brightnesses of the two windows are related by the affine transform $I' = \lambda I + \mu$ for some constants λ and μ with $\lambda > 0$.

6. Prove the equivalence of correlation and sum of squared differences for images with zero mean and unit Frobenius norm.

7. Recursive computation of the correlation function:

   (a) Show that

   $$(\boldsymbol{w} - \bar{\boldsymbol{w}}) \cdot (\boldsymbol{w}' - \bar{\boldsymbol{w}}') = \boldsymbol{w} \cdot \boldsymbol{w}' - (2m+1)(2n+1)\bar{I}\bar{I}'.$$

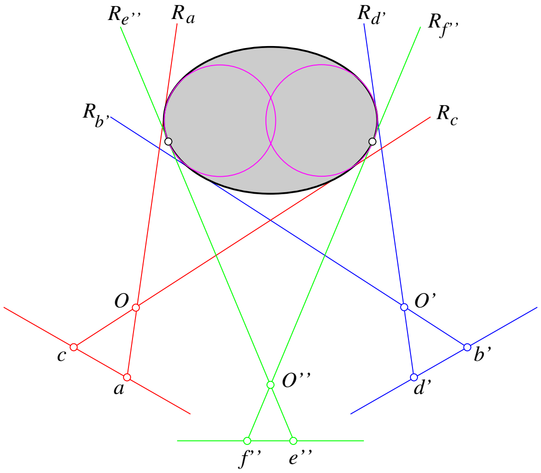
   (b) Show that the average intensity $\bar{I}$ can be computed recursively, and estimate the cost of the incremental computation.

   (c) Generalize the above calculations to all elements involved in the construction of the correlation function, and estimate the overall cost of correlation over a pair of images.

8. Show how a first-order expansion of the disparity function for rectified images can be used to warp the window of the right image corresponding to a rectangular region of the left one. Show how to compute correlation in this case using interpolation to estimate right-image values at the locations corresponding to the centers of the left window's pixels.

9. Show how to predict curvature in one image from curvature measurements in two other pictures.

10. Three-camera reconstruction of smooth surfaces' occluding contours: show that, in the planar case, three matching rays provide enough constraints to reconstruct the circle of curvature as shown below.



Programming Assignments

1. Implement the rectification process.

2. Implement the algorithm developed in Exercise 4 for generating single-image random dot stereograms.

3. Implement a correlation-based approach to stereopsis.

4. Implement a multi-scale approach to stereopsis.

5. Implement a dynamic-programming approach to stereopsis.

6. Implement a trinocular approach to stereopsis.

Chapter 13

AFFINE STRUCTURE FROM MOTION

In this chapter we address the problem of recovering the three-dimensional structure of a scene from a sequence of pictures. We will suppose that n points have been observed in m images and that the correct correspondences between the features observed in successive images have been established (through tracking for example). We will also assume an affine projection model: in other words, the observed objects undergo an affine motion/deformation before being projected orthographically onto the picture plane.

Following Koenderink and Van Doorn [?], we stratify the solution to this problem into two phases: (1) First use at least two views of the scene and purely affine measurements (e.g., ratios of distances) to construct a unique (up to an arbitrary affine transformation) three-dimensional representation of the scene; (2) use additional views and metric measurements (distances or angles) to uniquely determine the rigid structure of the scene.

As argued by Koenderink and Van Doorn, the first step yields the essential part of the solution: the affine structure is a full three-dimensional representation of the scene, which can be used in its own right to construct new views of the scene, or, as shown in Section 13.5, to segment the data points into objects undergoing different motions [?; ?; ?]. The second step simply amounts to finding a single affine transformation that will account for the rigidity of the scene and align it with the correct Euclidean frame. In addition, purely affine methods do not require camera calibration since the corresponding transformation of the image coordinates can be folded into the overall affine deformation of the object. This may prove useful for active vision systems whose calibration parameters vary dynamically, or, for example, for planetary robot probes whose parameters may have been altered by the large accelerations at take-off and landing. As shown in Section 13.4, recovering the Euclidean structure from the affine one requires, however, knowing at least some of the calibration parameters (e.g., the aspect ratio of the pixels).

Additional arguments in favor of a stratification of three-dimensional motion analysis have been advanced by Faugeras [?] in the more general setting of central projection models, but we will focus our attention on affine models in this chapter. This is not overly restrictive for small fields of view and restricted depth ranges. As shown in Section 13.4.1, the affine projection model subsumes three well known approximations to full perspective projection: orthography, weak perspective, and paraperspective [**?**], and it can account for variations in depth as well as certain types of perspective distortions.

Like Ullman's classical work on (Euclidean) shape from motion [**?**], the affine structure-from-motion method of Koenderink and Van Doorn is concerned with shape recovery from a *minimum* number of images. Using more images overconstrains the problem and leads to more robust least-squares solutions. Accordingly, the second part of this chapter is devoted to the problem of recovering the affine shape of a scene from several (possibly many) pictures. In particular, we will elucidate the structure of affine images, showing it to be the key to powerful linear solutions to this problem [**?**; **?**].

13.1 Elements of Affine Geometry

As noted in [**?**], affine geometry is, roughly speaking, what remains after practically all ability to measure length, area, angles, etc.. has been removed from Euclidean geometry. The concept of parallelism remains, however, as well as the concept of affine transformations, i.e., bijections that preserve parallelism and the ratio of distances between collinear points.

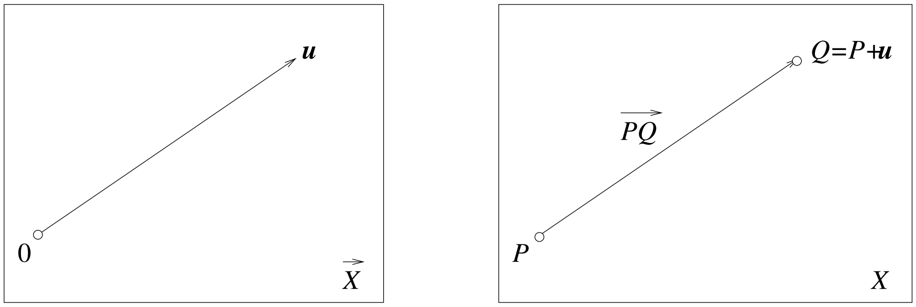
Giving a rigorous axiomatic introduction to affine geometry would be out of place here. Instead, we will remain quite informal, and just recall the basic facts about real affine spaces that are necessary to understand the rest of this chapter. The reader familiar with notions such as barycentric combinations, affine coordinate systems, and affine transformations may safely proceed to the next section.

Roughly speaking once again, a real affine space is a set X of *points*, together with a real vector space $\vec{X}$, and an *action* ϕ of the additive group of V on X. The vector space $\vec{X}$ is said to *underlie* the affine space X. Informally, the action of a group on a set maps the elements of this group onto bijections of the set. Here, the action ϕ associates with every vector $\boldsymbol{u} \in \vec{X}$ a bijection $\phi_{\boldsymbol{u}} : X \to X$ such that, for any $\boldsymbol{u}, \boldsymbol{v}$ in $\vec{X}$ and any point P in X, $\phi_{\boldsymbol{u}+\boldsymbol{v}}(P) = \phi_{\boldsymbol{u}}(\phi_{\boldsymbol{v}}(P))$, $\phi_0(P) = P$, and for any pair of points P, Q in X, there exists a unique vector $\boldsymbol{u}$ in $\vec{X}$ such that $\phi_{\boldsymbol{u}}(P) = Q$.

Although the points of an affine space cannot be "added", they can be "subtracted", i.e., $Q - P \stackrel{\text{def}}{=} \overrightarrow{PQ}$, and the *barycentric combination* (or *affine combination*) of $p+1$ points can also be defined as follows: Consider the points $A_0, \dots, A_m$ and $m+1$ real weights $\alpha_0, \dots, \alpha_m$ such that $\alpha_0 + \ldots + \alpha_m = 1$; the corresponding barycentric combination is the point

$$\sum_{i=0}^{m} \alpha_i A_i \stackrel{\text{def}}{=} A_j + \sum_{i=0, i\neq j}^{m} \alpha_i (A_i - A_j),$$

A familiar affine space is of course the physical three-dimensional space, where X is the set of physical points, and $\vec{X}$ is the set of translations of X onto itself. Another affine space of interest can be constructed by choosing both X and $\vec{X}$ to be equal to $\mathbb{R}^n$, with the action ϕ defined by $\phi_{\boldsymbol{u}}(P) = P + \boldsymbol{u}$, where P and $\boldsymbol{u}$ are both elements of $\mathbb{R}^n$ and "+" denotes the addition in $\mathbb{R}^n$.



These two examples justify the notation that will be used in the rest of this chapter: we will usually denote the point $\phi_{\boldsymbol{u}}(P)$ by $P + \boldsymbol{u}$ and the vector $\boldsymbol{u}$ such that $\phi_{\boldsymbol{u}}(P) = Q$ by $\overrightarrow{PQ}$.

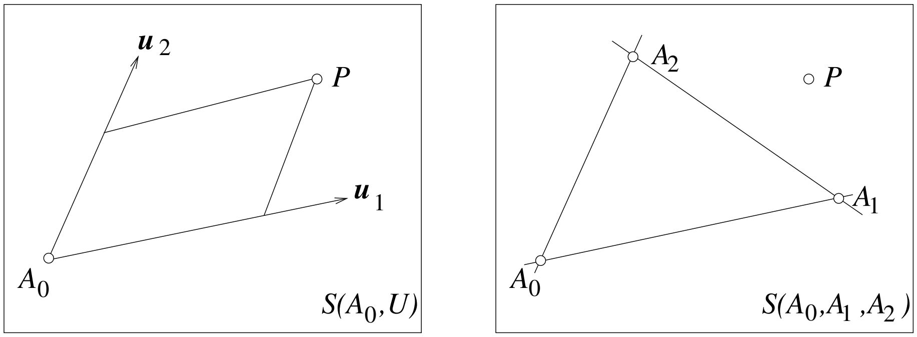
Example 13.1: *Affine examples*

where j is an integer between 0 and m. It is easily verified that this point is independent of the value of j (it is of course essential that the weights α_i add to 1 for this definition to make sense).

An affine subspace is defined by a point O of X and a vector subspace U of $\vec{X}$ as the set of points $S(O, U) \stackrel{\text{def}}{=} \{O + \boldsymbol{u}, \boldsymbol{u} \in U\}$. The dimension of an affine subspace is the dimension of the associated vector subspace. Two affine subspaces associated with the same vector subspace are said to be parallel. Barycentric combinations can be used to define affine subspaces purely in terms of points: The subspace spanned by the points $A_0, \ldots, A_m$ is the set $S(A_0, \ldots, A_m) \stackrel{\text{def}}{=} \{\sum_{i=0}^m \alpha_i A_i | \sum_{i=0}^m \alpha_i = 1\}$ of all barycentric combinations of these points. It is easy to verify that $S(A_0, \ldots, A_m)$ is indeed an affine subspace, and that its dimension is at most m (e.g., two distinct points define a line, three points define (in general) a plane etc.). We will say that $m + 1$ points are independent if they do not lie in a subspace of dimension at most $m - 1$, so $m + 1$ independent points define an m-dimensional subspace.

An affine coordinate system for $S(O, U)$ consists of a point A_0 (called the origin of the coordinate system) in $S(O, U)$ and a coordinate system $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$ for U. The affine coordinates of the point $P \in S(O, U)$ are defined as the coordinates of the vector $\overrightarrow{A_0P}$ in the coordinate system $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$. An alternative way of defining a coordinate system for an affine subspace Y is to pick $m + 1$ independent points $A_0, \ldots, A_m$ in Y. The barycentric coordinates α_i $(i = 0, \ldots, m)$ of a point

Consider three non-collinear points A_0, A_1 and A_2 in $\mathbb{R}^3$ considered as an affine space. These points span the plane $S(A_0, A_1, A_2)$, and any point P in that plane can be represented as a barycentric combination of these three points.



Equivalently, the plane can be viewed as the affine subspace $S(A_0, U)$ of $\mathbb{R}^3$ associated with the point A_0 and the vector plane U spanned by the two vectors $\boldsymbol{u}_1 = \overrightarrow{A_0A_1}$ and $\boldsymbol{u}_2 = \overrightarrow{A_0A_2}$.

Example 13.2: *More affine examples.*

P in Y are uniquely defined by $P = \alpha_0 A_0 + \ldots + \alpha_m A_m$. Note that the barycentric coordinates of the basis points are $(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)$.

When an n-dimensional affine space X has been equipped with an affine basis, a necessary and sufficient condition for $m+1$ points A_i to define a p-dimensional affine subspace of X (with $m \geq p$ and $n \geq p$) is that the $(n+1) \times (m+1)$ matrix

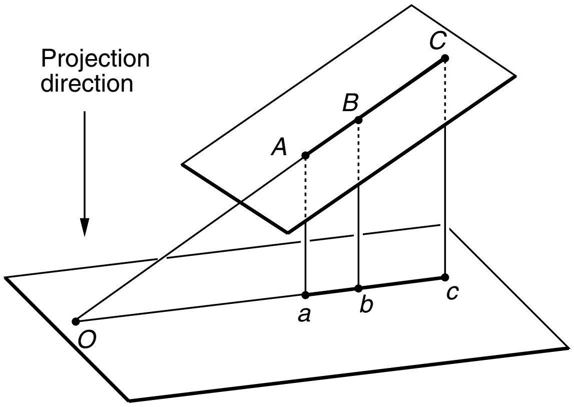
$$\begin{pmatrix} x_{00} & \ldots & x_{m0} \\ \ldots & \ldots & \ldots \\ x_{0n} & \ldots & x_{mn} \\ 1 & \ldots & 1 \end{pmatrix}$$

formed by the coordinate vectors $(x_{i0}, \ldots, x_{in})^T$ $(i = 0, \ldots, m)$ has rank $p+1$: indeed, a rank lower than $p+1$ means that any column of this matrix is a barycentric combination of at most p of its columns, and a rank higher than $p+1$ implies that at least $p+2$ of the points are independent.

An affine transformation between two affine subspaces X and Y is a bijection from X onto Y that preserves parallelism and affine coordinates. Equivalently, affine transformations can be characterized by the fact that they preserve the ratio of signed distances along parallel oriented lines.

Finally, the relationship between vector spaces and affine spaces induces a relationship between linear and affine transformations. In particular, it is easily shown that an affine transformation $\psi : X \to Y$ between two affine subspaces X and Y

Parallel projections from a plane onto another one are affine transformations: the triangles OAa, OBb and OCc are similar, and it follows that $\overline{AB}/\overline{BC} = \overline{ab}/\overline{bc}$ for any orientation of the lines OC and Oc. The parallelism of lines is obviously preserved by parallel projection.



This property of parallel projection will play an essential role in the next section.

Example 13.3: *Parallel projection.*

associated with the vector spaces $\vec{X}$ and $\vec{Y}$ can be written as

$$\psi(P) = \psi(O) + \vec{\psi}(P - O),$$

where O is some arbitrarily chosen origin, and $\vec{\psi} : \vec{X} \rightarrow \vec{Y}$ is a linear mapping from $\vec{X}$ onto $\vec{Y}$ that is independent of the choice of O. When X and Y are of (finite) dimension m and an affine coordinate system with origin O is chosen, this yields the familiar expression

$$\psi(\boldsymbol{P}) = \boldsymbol{t} + \mathcal{A}\boldsymbol{P},$$

where $\boldsymbol{P}$ denotes the coordinate vector of P in the chosen basis, $\boldsymbol{t}$ denotes the coordinate vector of $\psi(O)$, and $\mathcal{A}$ is the $m \times m$ matrix representing $\vec{\psi}$ in the same coordinate system.

13.2 Affine Structure from Two Images

We now have the right tools for estimating the three-dimensional structure of a scene from two images. We follow Koenderink and Van Doorn [**?**] and first solve for the affine structure, before taking into account the rigidity constraints and metric measurements that will map the affine structure onto a Euclidean one.

13.2.1 The Affine Structure-from-Motion Theorem

Given two orthographic images of five points A, B, C, D and P, is it possible to reconstruct the affine coordinates of P in the basis (A, B, C, D)?

Koenderink and Van Doorn [**?**] have shown that the answer to this question is positive, exploiting the fact that the orthographic projection of a plane onto another plane is an affine transformation. In particular, when the point P belongs to the plane Π that contains the triangle ABC, its affine coordinates in the basis of Π formed by these three points can be directly measured in either of the two images.

Now let E (resp. Q) denote the intersection of the line passing through the points D and d' (resp. P and p') with the plane Π (Figure 13.1). The projections e'' and q'' of the points E and P onto the plane Π'' have the same affine coordinates in the basis (a'', b'', c'') as the points d' and p' in the basis (a', b', c').

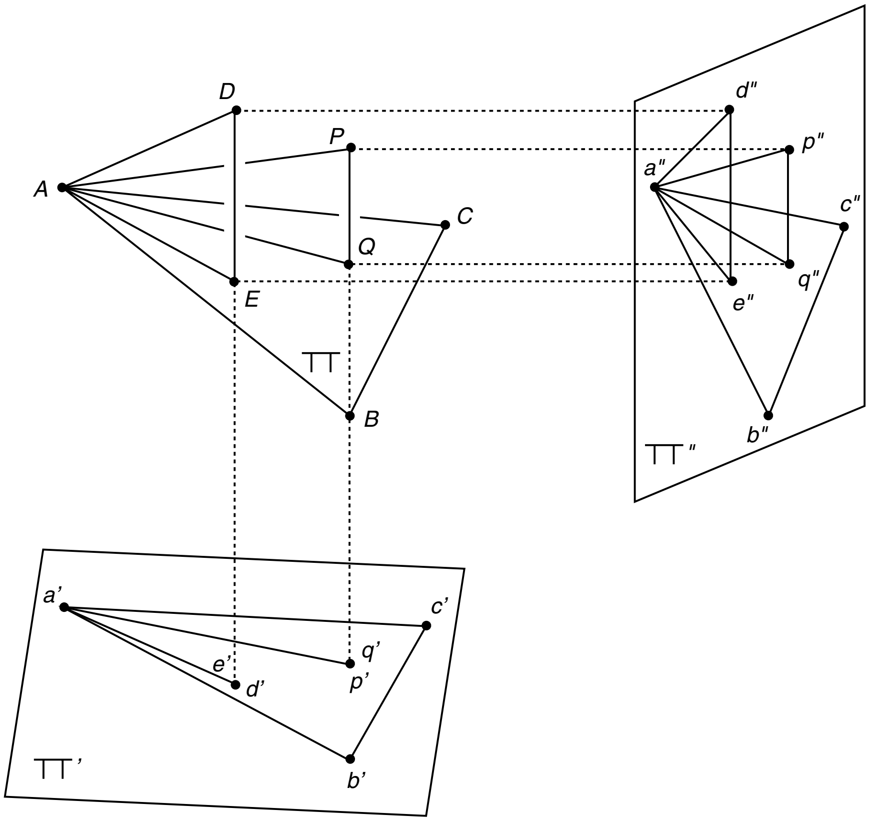


Figure 13.1. Geometric construction of the affine coordinates of a point P in the basis formed by the four points A, B, C and D.

In addition, since the two segments ED and QP are parallel to the first projection direction, the two line segments $e''d''$ and $q''p''$ are also parallel, and we can measure the ratio

$$\lambda = \frac{\overline{q''p''}}{\overline{e''d''}} = \frac{\overline{QP}}{\overline{ED}},$$

where $\overline{AB}$ denotes the signed distance between the two points A and B for some arbitrary (but fixed) orientation of the line joining these points.

If we now denote by $(\alpha_{d'}, \beta_{d'})$ and $(\alpha_{p'}, \beta_{p'})$ the coordinates of the points $d' = e'$ and $p' = q'$ in the basis (a', b', c'), we can write

$$\begin{aligned}\overrightarrow{AP} &= \overrightarrow{AQ} + \overrightarrow{QP} \\ &= \alpha_{p'}\overrightarrow{AB} + \beta_{p'}\overrightarrow{AC} + \lambda\overrightarrow{ED} \\ &= (\alpha_{p'} - \lambda\alpha_{d'})\overrightarrow{AB} + (\beta_{p'} - \lambda\beta_{d'})\overrightarrow{AC} + \lambda\overrightarrow{AD}.\end{aligned}$$

In other words, the affine coordinates of P in the (A, B, C, D) basis are $(\alpha_{p'} - \lambda\alpha_{d'}, \beta_{p'} - \lambda\beta_{d'}, \lambda)$. This is the *affine structure-from-motion theorem*: given two orthographic views of four non-coplanar points, the affine structure of the scene is uniquely determined [**?**].

Figure 13.2 shows three projections of the synthetic face used in Koenderink's and Van Doorn's experiments, along with an affine profile view computed from two of the images.

13.2.2 Rigidity and Metric Constraints

When the observed object is rigid, the transformation between the two views goes from affine to Euclidean, i.e., it is the composition of a rotation and a translation. Under orthographic projection, a translation in depth has no effect, and a translation in the image plane (fronto-parallel translation) is easily eliminated by aligning the two projections of the point A. Any rotation about the viewing direction is also easily identified and discarded [**?**]. At this stage, the two views differ by a rotation about some axis in a fronto-parallel plane passing through the projection of A, and Koenderink and Van Doorn show that there exists a one-parameter family of such rotations, determining the shape up to a depth scaling and a shear. The addition of a third view finally restricts the solution to one or two pairs related through a reflection in the fronto-parallel plane. This construction is a bit too involved to be included here. Instead, we will detail in Section 13.4 the passage from affine to Euclidean structure in the multi-image case.

Figure 13.3 shows a profile view of the Euclidean face reconstructed from the three views of Figure 13.2.

13.3 Affine Structure from Multiple Images

The method presented in the previous section is aimed at recovering the affine scene structure from a minimum number of images. We now address the problem of estimating the same information from a potentially large number of pictures, and

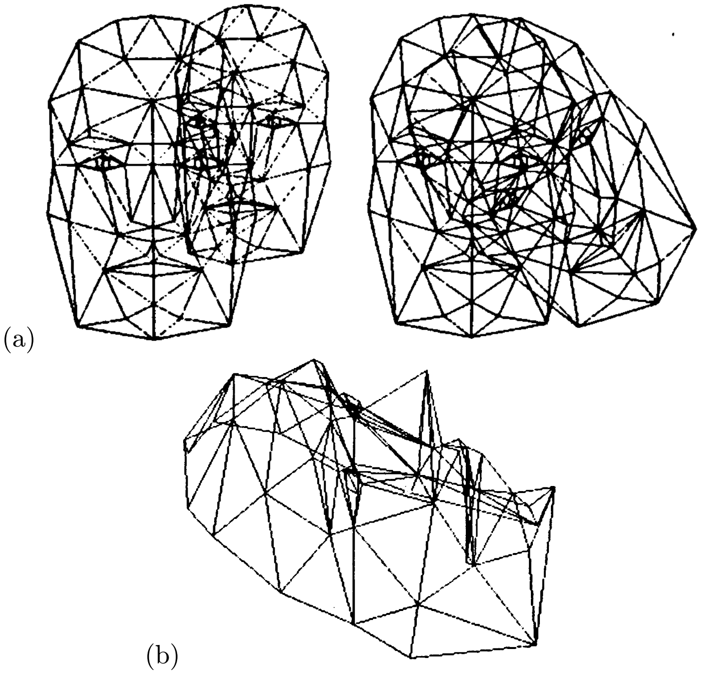


Figure 13.2. Affine reconstruction from two views – experimental results: (a) three views of a face: views 0 and 1 are overlaid on the left, and views 1 and 2 are overlaid on the right; (b) profile view of the affine reconstruction, computed from images 0 and 1. Reprinted from [**?**], Figures 1 and 6.

switch from a mostly geometric approach to an algebraic one. The set of affine images of a scene is first shown to also exhibit an affine structure, which is then exploited to derive the factorization method of Tomasi and Kanade [**?**] for estimating the affine structure and motion of a scene from an image sequence.[1]

13.3.1 The Affine Structure of Affine Image Sequences

Let us consider an affine camera observing some three-dimensional object, i.e., let us assume that the scene, represented in some fixed affine coordinate system, is first submitted to an affine transformation and then orthographically projected onto the image plane of the camera.

We will suppose that we observe a point P with a fixed set of m cameras and

---

[1] This method was originally proposed by Tomasi and Kanade as an approach to *Euclidean* structure and motion recovery from orthographic views. Indeed, Section 13.4 will show that the Euclidean structure and motion are easily computed from the affine ones. However, in keeping with Koenderink's and Van Doorn's view of a stratified approach to motion analysis, we believe that Tomasi's and Kanade's method is better understood in the setting of a two-phase process whose main step is the affine one.

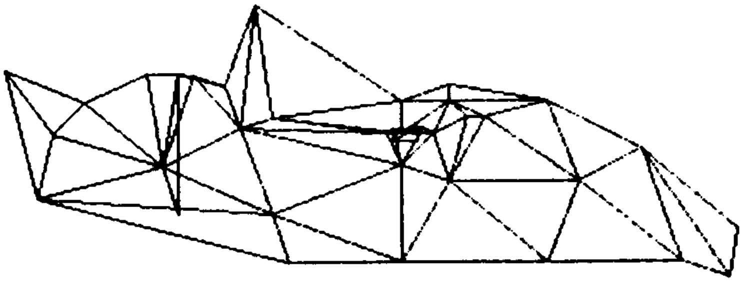


Figure 13.3. Euclidean reconstruction from the three views of a face shown in Figure 13.2. Reprinted from [?], Figure 9.

denote by p_i $(i = 1, \dots, m)$ the corresponding image points. Let us also denote the coordinate vector of P in the object coordinate system by $\boldsymbol{P} = (x, y, z)^T$ and use $\boldsymbol{p}_i = (u_i, v_i)^T$ to denote the coordinate vector of p_i. The affine camera model can now be written as

$$\boldsymbol{p}_i = \boldsymbol{o} + \mathcal{M}_i \boldsymbol{P}, \tag{13.3.1}$$

where $\mathcal{M}_i$ is a 2×3 matrix and $\boldsymbol{o}$ is the position of the projection into the image of the object coordinate system's origin.

Stacking the m instances of (13.3.1) yields

$$\boldsymbol{d} = \boldsymbol{r} + \mathcal{M}\boldsymbol{P},$$

where

$$\boldsymbol{d} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{p}_1 \\ \dots \\ \boldsymbol{p}_m \end{pmatrix}, \quad \boldsymbol{r} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{o}_1 \\ \dots \\ \boldsymbol{o}_m \end{pmatrix} \quad \text{and} \quad \mathcal{M} \stackrel{\text{def}}{=} \begin{pmatrix} \mathcal{M}_1 \\ \dots \\ \mathcal{M}_m \end{pmatrix}.$$

This shows that the set of images taken by the cameras is the three-dimensional affine subspace of $\mathbb{R}^{2m}$ spanned by the point $\boldsymbol{r}$ and the column vectors of the $2m \times 3$ matrix $\mathcal{M}$.

In particular, if we now consider n points P_i observed by m cameras, we can now define a $(2m+1) \times n$ data matrix

$$\begin{pmatrix} \boldsymbol{d}_1 & \dots & \boldsymbol{d}_n \\ 1 & \dots & 1 \end{pmatrix},$$

and it follows from Section 13.1 that this matrix has rank 4.

13.3.2 A Factorization Approach to Affine Motion Analysis

Tomasi and Kanade [?] have exploited the affine structure of affine images in a robust factorization method for estimating the structure of a scene and the corresponding camera motion through singular value decomposition [?] (see insert).

Technique: Singular Value Decomposition

Let $\mathcal{A}$ be an $m \times n$ matrix, with $m \geq n$, then $\mathcal{A}$ can always be written as

$$\mathcal{A} = \mathcal{U}\mathcal{W}\mathcal{V}^T,$$

where:

- $\mathcal{U}$ is an $m \times n$ column-orthogonal matrix, i.e., $\mathcal{U}^T\mathcal{U} = \text{Id}_m$,
- $\mathcal{W}$ is a diagonal matrix whose diagonal entries w_i $(i = 1, \ldots, n)$ are the singular values of $\mathcal{A}$ with $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$,
- and $\mathcal{V}$ is an $n \times n$ orthogonal matrix, i.e., $\mathcal{V}^T\mathcal{V} = \mathcal{V}\mathcal{V}^T = \text{Id}_n$.

This is the *singular value decomposition* (SVD) of the matrix $\mathcal{A}$, and it can be computed using the algorithm described in [**?**].
Suppose now that $\mathcal{A}$ has rank $p < n$, then the matrices $\mathcal{U}$, $\mathcal{W}$, and $\mathcal{V}$ can be written as

$$\mathcal{U} = \begin{array}{|c|c|}\hline \mathcal{U}_p & \mathcal{U}_{n-p} \\ \hline \end{array} \quad \mathcal{W} = \begin{array}{|c|c|}\hline \mathcal{W}_p & 0 \\ \hline 0 & 0 \\ \hline \end{array} \quad \text{and} \quad \mathcal{V}^T = \begin{array}{|c|}\hline \mathcal{V}_p^T \\ \hline \mathcal{V}_{n-p}^T \\ \hline \end{array},$$

and

- the columns of $\mathcal{U}_p$ form an orthonormal basis of the space spanned by the columns of $\mathcal{A}$, i.e., its *range*,
- and the columns of $\mathcal{V}_{n-p}$ for a basis of the space spanned by the solutions of $A\boldsymbol{x} = 0$, i.e., the *null space* of this matrix.

Both $\mathcal{U}_p$ and $\mathcal{V}_p$ are $n \times p$ column-orthogonal matrices, and we have of course $\mathcal{A} = \mathcal{U}_p\mathcal{W}_p\mathcal{V}_p^T$. The following two theorems show that singular value decomposition also provides a valuable *approximation* procedure. In both cases, $\mathcal{U}_p$ and $\mathcal{V}_p$ denote as before the matrices formed by the p leftmost columns of the matrices $\mathcal{U}$ and $\mathcal{V}$, and $\mathcal{W}_p$ is the $p \times p$ diagonal matrix formed by the p largest singular values. This time, however, $\mathcal{A}$ may have maximal rank n, and the remaining singular values may be nonzero.

Theorem 3: *When $\mathcal{A}$ has a rank greater than p, $\mathcal{U}_p\mathcal{W}_p\mathcal{V}_p^T$ is the best possible rank-p approximation of $\mathcal{A}$ (in the sense of the* Frobenius norm, *i.e., the norm induced on matrices by the Euclidean vector norm).*

Theorem 4: *Let $\boldsymbol{a}_i \in \mathbb{R}^m$ $(i = 1, \ldots, n)$ denote the n column vectors of the matrix $\mathcal{A}$; the vector subspace V_p of dimension p that minimizes the mean squared error*

$$\frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{a}_i - \boldsymbol{b}_i|^2,$$

where $\boldsymbol{b}_i$ denotes the orthogonal projection of $\boldsymbol{a}_i$ into V_p, is the subspace of $\mathbb{R}^m$ spanned by the columns of $\mathcal{U}_p$.

These two theorems will be used repeatedly in the rest of this book. Singular value decomposition has other important properties, for example:

- The SVD of a matrix can be used to determine its rank numerically.
- Singular value decomposition can be used to compute the solution of a linear least-squares problem.

Assuming that the origin of the object coordinate system is one of the observed points or their center of mass, we can translate the origin of the image coordinate system to the corresponding image point. Under this transformation, the set of images of a scene becomes the three-dimensional *vector space* defined by

$$\boldsymbol{p}_i = \mathcal{M}_i \boldsymbol{P} \quad \text{for} \quad i = 1, \ldots, m \quad \text{or equivalently} \quad \boldsymbol{d} = \mathcal{M}\boldsymbol{P}. \tag{13.3.2}$$

We can now define the $2m \times n$ data matrix

$$\mathcal{D} \stackrel{\text{def}}{=} (\,\boldsymbol{d}_1 \quad \ldots \quad \boldsymbol{d}_n\,) = \mathcal{M}\mathcal{P}, \quad \text{with} \quad \mathcal{P} \stackrel{\text{def}}{=} (\,\boldsymbol{P}_1 \quad \ldots \quad \boldsymbol{P}_n\,)\,.$$

As the product of a $2m \times 3$ matrix and a $3 \times n$ matrix, $\mathcal{D}$ has, in general, rank 3. If $\mathcal{U}\mathcal{W}\mathcal{V}^T$ is the singular value decomposition, this means that only three of the singular values are nonzero, thus $\mathcal{D} = \mathcal{U}_3\mathcal{W}_3\mathcal{V}_3^T$, where $\mathcal{U}_3$ and $\mathcal{V}_3$ denote the $2m \times 3$ and $3 \times n$ matrices formed by the three leftmost columns of the matrices $\mathcal{U}$ and $\mathcal{V}$, and $\mathcal{W}_3$ is the 3×3 diagonal matrix formed by the corresponding nonzero singular values.

We claim that we can take $\mathcal{M} = \mathcal{U}_3$ and $\mathcal{P} = \mathcal{W}_3\mathcal{V}_3^T$ are representative of the true *affine* camera motion and scene structure. Indeed, the columns of $\mathcal{M}$ form by definition a basis for the range of $\mathcal{D}$. The columns of $\mathcal{U}_3$ form by construction another basis for this range. This implies that there exists a 3×3 matrix $\mathcal{Q}$ such that $\mathcal{M} = \mathcal{U}_3\mathcal{Q}$ and thus $\mathcal{P} = \mathcal{Q}^{-1}\mathcal{W}_3\mathcal{V}_3^T$. Conversely, $\mathcal{D}_3 = (\mathcal{U}_3\mathcal{Q})(\mathcal{Q}^{-1}\mathcal{W}_3\mathcal{V}_3^T)$ for any invertible 3×3 matrix $\mathcal{Q}$. Since the origin of the world coordinate system can of course be set arbitrarily, it follows that the structure and motion can only be recovered *up to an affine transformation*, and singular value decomposition indeed outputs a representant of the affine motion and scene structure.

Our reasoning so far is of course only valid in an idealized, noiseless case. In practice, due to image noise, errors in localization of feature points, and to the mere fact that actual cameras are not affine, (13.3.2) will not hold exactly and the matrix $\mathcal{D}$ will have (in general) full rank. Let us show that singular value decomposition still yields a reasonable estimate of the affine structure and motion in this case. Since (13.3.2) does not hold exactly, the best we can hope for is to minimize

$$E \stackrel{\text{def}}{=} \sum_{i,j} |\boldsymbol{p}_{ij} - \mathcal{M}_i\boldsymbol{P}_j|^2 = \sum_j |\boldsymbol{d}_j - \mathcal{M}\boldsymbol{P}_j|^2.$$

with respect to the matrices $\mathcal{M}_i$ $(i = 1, \ldots, m)$ and vectors $\boldsymbol{P}_j$ $(j = 1, \ldots, m)$.

Writing that the partial derivative of E with respect to $\boldsymbol{P}_j$ should be zero at a minimum yields

$$0 = \frac{\partial E}{\partial \boldsymbol{P}_j} = [2\frac{\partial}{\partial \boldsymbol{P}_j}(\boldsymbol{d}_j - \mathcal{M}\boldsymbol{P}_j)^T][\boldsymbol{d}_j - \mathcal{M}\boldsymbol{P}_j] = -2\mathcal{M}^T[\boldsymbol{d}_j - \mathcal{M}\boldsymbol{P}_j],$$

thus

$$\boldsymbol{P}_j = \mathcal{M}^\dagger \boldsymbol{d}_j,$$

where $\mathcal{M}^\dagger \overset{\text{def}}{=} (\mathcal{M}^T\mathcal{M})^{-1}\mathcal{M}^T$ is the pseudoinverse of $\mathcal{M}$. Substituting back into E shows that the minimum value of E verifies

$$E = \sum_j |(\text{Id} - \mathcal{M}\mathcal{M}^\dagger)\boldsymbol{d}_j|^2.$$

Now observe that the matrix $\mathcal{M}\mathcal{M}^\dagger$ associates with any vector $\boldsymbol{d}$ in $\mathbb{R}^{2m}$ its orthogonal projection onto the three-dimensional subspace $V_{\mathcal{M}}$ spanned by the columns of $\mathcal{M}$ (see exercises). It follows that E measures the mean squared norm of the difference between the vectors $\boldsymbol{d}_j$ and their orthogonal projections onto $V_{\mathcal{M}}$. According to Theorem 4, E is minimum when $V_{\mathcal{M}}$ is the range of the matrix $\mathcal{U}_3$ formed by the three leftmost columns of $\mathcal{U}$, where $\mathcal{U}\mathcal{W}\mathcal{V}^T$ denotes as before the singular value decomposition of $\mathcal{D}$. In particular, the matrix $\mathcal{M}$ minimizing E verifies $V_{\mathcal{M}} = V_{\mathcal{U}_3}$ and we can take $\mathcal{M} = \mathcal{U}_3$.

As noted earlier, the corresponding estimate of the point position is $\boldsymbol{P}_j = \mathcal{M}^\dagger\boldsymbol{d}_j$, thus, since $\mathcal{U}_3$ is column-orthogonal,

$$\mathcal{P} = \mathcal{M}^\dagger\mathcal{D} = [(\mathcal{U}_3^T\mathcal{U}_3)^{-1}\mathcal{U}_3^T][\mathcal{U}\mathcal{W}\mathcal{V}^T] = \mathcal{W}_3\mathcal{V}_3^T,$$

where, as before, $\mathcal{V}_3$ denotes the $3 \times n$ matrix formed by the three leftmost columns of the matrix $\mathcal{V}$, and $\mathcal{W}_3$ is the 3×3 diagonal matrix formed by the corresponding singular values. In particular, singular value decomposition can be used to estimate the affine structure and motion from the data matrix $\mathcal{D}$, as shown in Algorithm 13.1.

1. Compute the singular value decomposition $\mathcal{D} = \mathcal{U}\mathcal{W}\mathcal{V}^T$.
2. Construct the matrices $\mathcal{U}_3$, $\mathcal{V}_3$, and $\mathcal{W}_3$ formed by the three leftmost columns of the matrices $\mathcal{U}$ and $\mathcal{V}$, and the corresponding 3×3 sub-matrix of $\mathcal{W}$.
3. Define
$$\mathcal{M} = \mathcal{U}_3 \quad \text{and} \quad \mathcal{P} = \mathcal{W}_3\mathcal{V}_3^T;$$
the $2m \times 3$ matrix $\mathcal{M}$ is an estimate of the camera motion, and the $3 \times n$ matrix $\mathcal{P}$ is an estimate of the scene structure.

Algorithm 13.1: *Tomasi's and Kanade's factorization algorithm for affine shape from motion.*

13.4 From Affine to Euclidean Images

As noted in Section 13.2, taking rigidity into account allows the recovery of Euclidean shape from three orthographic views. Here we address a similar problem in the case where more than three views are available, and consider several Euclidean projection models subsumed by the affine projection model.

13.4.1 Euclidean Projection Models

We examine the orthographic, weak perspective (scaled orthographic), and paraperspective [**?**] projection models (Figure 13.4), and assume that the camera observing the scene has been calibrated so that image points are represented by their normalized coordinate vectors.[2] We shall see that the affine projection equation (13.3.1) still holds for these models. However, this time there are some constraints on the components of the projection matrix $\mathcal{M}$.

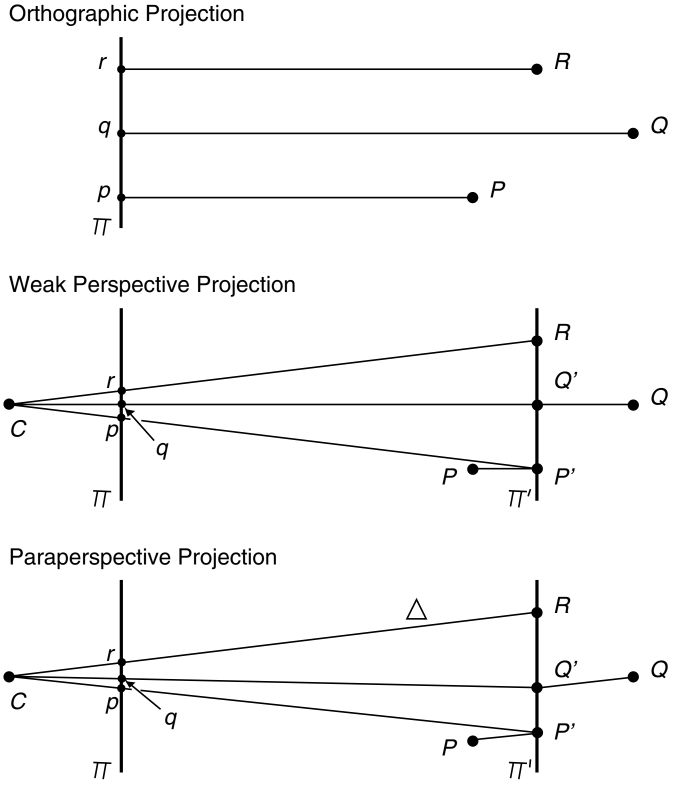


Figure 13.4. Three projection models.

For an orthographic camera, the matrix $\mathcal{M}$ is part of a rotation matrix, and its rows $\boldsymbol{a}^T$ and $\boldsymbol{b}^T$ are unit vectors orthogonal to each other. In other words, an orthographic camera is an affine camera with the additional constraints

$$\boldsymbol{a} \cdot \boldsymbol{b} = 0 \quad \text{and} \quad |\boldsymbol{a}|^2 = |\boldsymbol{b}|^2 = 1. \tag{13.4.1}$$

---

[2] Strictly speaking, this is not completely necessary: the orthographic and weak perspective models only require that the camera aspect ratio be known. However, the paraperspective projection model utilizes the absolute image position of a reference point, and this requires that the image center be known.

Weak perspective is an approximation of perspective projection, constructed as follows (Figure 13.4): Let C denote the optical center of the camera and let R denote a scene reference point; the weak perspective projection of a scene point P is constructed in two steps: P is first projected orthographically onto a point P' of the plane Π' parallel to the image plane Π and passing through R. Perspective projection is then used to map the point P' onto the image point p. Since Π' is a fronto-parallel plane, the net effect of the second projection step is a scaling of the image coordinates, and it follows that a weak perspective camera is an affine camera with the two constraints

$$\boldsymbol{a} \cdot \boldsymbol{b} = 0 \quad \text{and} \quad |\boldsymbol{a}|^2 = |\boldsymbol{b}|^2. \tag{13.4.2}$$

Paraperspective projection [**?**] is a refined approximation of perspective that takes into account the distortions associated with a reference point that is off the optical axis of the camera (Figure 13.4). Using the same notation as before, and denoting by Δ the line joining the optical center C to the reference point R, parallel projection in the direction of Δ is first used to map P onto a point P' of the plane Π'; perspective projection is then used to map the point P' onto the image point p. It is easily shown (see [**?**] for example) that a paraperspective camera is an affine camera that satisfies the constraints

$$\boldsymbol{a} \cdot \boldsymbol{b} = \frac{u_r v_r}{2(1+u_r^2)}|\boldsymbol{a}|^2 + \frac{u_r v_r}{2(1+v_r^2)}|\boldsymbol{b}|^2 \quad \text{and} \quad (1+v_r^2)|\boldsymbol{a}|^2 = (1+u_r^2)|\boldsymbol{b}|^2, \tag{13.4.3}$$

where (u_r, v_r) denote the coordinates of the perspective projection of the point R. It should be noted that under this projection model, the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ do not form a basis of the image plane. Instead, they form a basis of the vector plane orthogonal to the line joining the optical center of the camera to the reference point.

As expected, the paraperspective constraints reduce to the weak perspective constraints when $u_r = v_r = 0$, and these reduce in turn to the orthographic constraints when the planes Π and Π' coincide.

13.4.2 From Affine to Euclidean Motion

Let us now show how to recover the Euclidean structure from the affine one under orthographic projection. Let $\mathcal{Q}$ denote the 3×3 matrix associated with the linear mapping between the affine shape $\mathcal{P}$ and motion $\mathcal{M}$ and their Euclidean counterparts $\hat{\mathcal{P}}$ and $\hat{\mathcal{M}}$ (of course $\mathcal{Q}$ is only defined up to an arbitrary rotation). As noted in Section 13.3.2, we must have $\hat{\mathcal{M}} = \mathcal{M}\mathcal{Q}$ and $\hat{\mathcal{P}} = \mathcal{Q}^{-1}\mathcal{P}$. The Euclidean constraints derived in the previous section can be used to compute $\mathcal{Q}$.

Assuming orthographic projection. we can rewrite the constraints (13.4.1) as

$$\begin{cases} \boldsymbol{a}_i^T \mathcal{Q}\mathcal{Q}^T \boldsymbol{b}_i = 0, \\ \boldsymbol{a}_i^T \mathcal{Q}\mathcal{Q}^T \boldsymbol{a}_i = 1, \\ \boldsymbol{b}_i^T \mathcal{Q}\mathcal{Q}^T \boldsymbol{b}_i = 1, \end{cases} \tag{13.4.4}$$

where $\boldsymbol{a}_i^T$ and $\boldsymbol{b}_i^T$ denote the rows of the matrix $\mathcal{M}_i$ for $i = 1, \dots, m$.

To determine $\mathcal{Q}$ uniquely, we can for example assume that

$$\hat{\mathcal{M}}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Together with the constraints obtained by writing (13.4.4) for the remaining $m-1$ images, we have 6 linear equations and $3m-3$ quadratic ones in the coefficients of $\mathcal{Q}$. Tomasi and Kanade [?] proposed solving these equations via non-linear least squares . An alternative is to consider (13.4.4) as a set of *linear* constraints on the matrix $\mathcal{R} \stackrel{\text{def}}{=} \mathcal{Q}\mathcal{Q}^T$. The coefficients of $\mathcal{R}$ are found in this case via linear least squares, and the coefficients of $\mathcal{Q}$ can then be computed via Cholesky decomposition. This is the method used in [?] for example (see [?] for another variant). It should be noted that it requires that the recovered matrix $\mathcal{B}$ be positive definite, which is not guaranteed in the presence of noise.

Figure 13.5 shows experimental results, including some input images, the corresponding feature tracks, and the recovered scene structure.

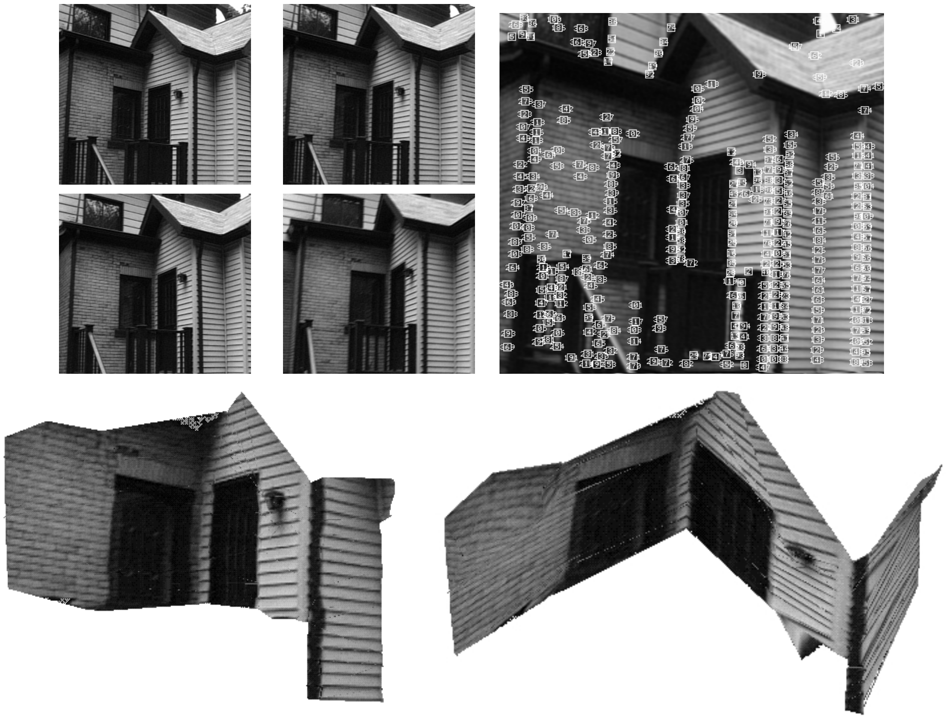


Figure 13.5. Euclidean shape from motion – experimental results: top-left: sample input images; top-right: the features automatically detected in the first frame; bottom: two views of the reconstructed scene. Reprinted from [?], Figures 6–7.

The weak and paraperspective cases can be treated in the same manner, except for the fact that the 2 constraints (13.4.2) or (13.4.3) written for $m-1$ images will replace the $3m-3$ constraints (13.4.4). See [?] for details.

13.5 Affine Motion Segmentation

We have assumed so far that the n points observed all undergo the same motion. What happens if these points belong instead to k objects undergoing different motions? We present two methods [?; ?] for segmenting the data points into such independently-moving objects (see [?] for another approach to the same problem).

13.5.1 The Reduced Echelon Form of the Data Matrix

Exactly as in Section 13.3.1, we can define the data matrix

$$\mathcal{D} = \begin{pmatrix} \boldsymbol{p}_{11} & \cdots & \boldsymbol{p}_{1n} \\ \cdots & \cdots & \cdots \\ \boldsymbol{p}_{m1} & \cdots & \boldsymbol{p}_{mn} \\ 1 & \ldots & 1 \end{pmatrix}.$$

This time, however, $\mathcal{D}$ does not have rank 4 anymore. Instead, the sub-matrices formed by the columns corresponding to each object will have rank 4 (or less), and the maximum rank of the overall data matrix will be $4k$. In other words, the columns of $\mathcal{D}$ corresponding to each object lie in four- (or less) dimensional subspaces of its range, and, as remarked by Gear [?], constructing the *reduced echelon form* of $\mathcal{D}$ will identify these subspaces and the column vectors that lie in them, providing a segmentation of the input points into rigid objects (or, to be more exact, into objects that may undergo affine deformations). Gear [?] gives several methods for computing the reduced echelon form using Gauss-Jordan elimination and QR reduction.

13.5.2 The Shape Interaction Matrix

The approach presented in the previous section relies only on the affine structure of affine images. Costeira and Kanade [?] have proposed a different method, based on a factorization of the data matrix. In the setting of motion segmentation, it is not possible to define a rank-3 data matrix for each object since the centroid of the corresponding points is unknown. Instead, Costeira and Kanade [?] construct, for $i=1,\ldots,k$, a rank-4 data matrix

$$\mathcal{D}^{(i)} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{p}_{11}^{(i)} & \cdots & \boldsymbol{p}_{1n_i}^{(i)} \\ \cdots & \cdots & \cdots \\ \boldsymbol{p}_{m1}^{(i)} & \cdots & \boldsymbol{p}_{mn_i}^{(i)} \end{pmatrix},$$

where n_i is the number of points associated with object number i. This matrix factorizes as $\mathcal{D}^{(i)} = \mathcal{M}^{(i)}\mathcal{P}^{(i)}$ where, this time,

$$\mathcal{M}^{(i)} \stackrel{\text{def}}{=} \begin{pmatrix} \mathcal{M}_1^{(i)} & \boldsymbol{o}_1^{(i)} \\ \dots & \dots \\ \mathcal{M}_m^{(i)} & \boldsymbol{o}_m^{(i)} \end{pmatrix} \quad \text{and} \quad \mathcal{P}^{(i)} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{P}_1^{(i)} & \dots & \boldsymbol{P}_{n_i}^{(i)} \\ 1 & \dots & 1 \end{pmatrix}.$$

Let us define the $2m \times n$ composite data matrix

$$\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{D}^{(1)}\mathcal{D}^{(2)} \dots \mathcal{D}^{(k)}),$$

as well as the composite $2m \times 4k$ (motion) and $4k \times n$ (structure) matrices

$$\mathcal{M} \stackrel{\text{def}}{=} (\mathcal{M}^{(1)}\mathcal{M}^{(2)} \dots \mathcal{M}^{(k)}) \quad \text{and} \quad \mathcal{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathcal{P}^{(1)} & 0 & \dots & 0 & 0 \\ 0 & \mathcal{P}^{(2)} & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \mathcal{P}^{(k)} \end{pmatrix}.$$

With this notation, we have

$$\mathcal{D} = \mathcal{M}\mathcal{P},$$

which confirms, of course, that $\mathcal{D}$ has rank $4k$ (or less).

As in the rank-3 case, the matrix $\mathcal{D}$ can be factorized using singular value decomposition, which also provides an estimate of its rank $r \leq 4k$. The best approximation $\mathcal{D}_r$ of rank r of $\mathcal{D}$ is then constructed and factorized as

$$\mathcal{D}_r = \mathcal{U}_r\mathcal{W}_r\mathcal{V}_r^T.$$

In the noiseless case, the columns of $\mathcal{D}$ and the columns of $\mathcal{U}_r$ span the same r-dimensional subspace of $\mathbb{R}^{2m}$, and there exists an $r \times r$ linear transformation between these two matrices. This implies in turn the existence of a non-singular $r \times r$ matrix B such that

$$\mathcal{V}_r^T = B\mathcal{P}.$$

Following Costeira and Kanade [**?**], we can now define the *shape interaction matrix* $\mathcal{Q}$ as

$$\mathcal{Q} \stackrel{\text{def}}{=} \mathcal{V}_r\mathcal{V}_r^T = \mathcal{P}^T(B^TB)\mathcal{P}.$$

Noting that the matrix B^TB is also a non-singular 4×4 matrix and that $\mathcal{V}_r$ is by construction column-orthogonal (i.e., $\mathcal{V}_r^T\mathcal{V}_r = \text{Id}_r$) now allows us to write

$$\mathcal{Q} = \mathcal{P}^T(B^{-1}B^{-T})^{-1}\mathcal{P} = P^T(B^{-1}\mathcal{V}_r^T\mathcal{V}_rB^{-T})\mathcal{P} = \mathcal{P}^T(\mathcal{P}\mathcal{P}^T)^{-1}\mathcal{P}.$$

The matrix $C \stackrel{\text{def}}{=} (\mathcal{P}\mathcal{P}^T)^{-1}$ is an $r \times r$ matrix, and it follows that the shape interaction matrix has the following block-diagonal structure

$$\mathcal{Q} = \begin{pmatrix} \mathcal{P}^{(1)T}C\mathcal{P}^{(1)} & 0 & \dots & 0 \\ 0 & \mathcal{P}^{(2)T}C\mathcal{P}^{(2)} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \mathcal{P}^{(k)T}C\mathcal{P}^{(k)} \end{pmatrix}.$$

The above construction assumes that the data points are ordered consistently with the object they belong to. In general, of course, this will not be the case. As remarked by Costeira and Kanade however, the values of the entries of the matrix $\mathcal{Q}$ are independent of the order of the points: changing this order will swap the rows and columns of $\mathcal{D}$ and $\mathcal{Q}$ in the same way. Thus recovering the correct point ordering (and the corresponding segmentation into objects) amounts to finding the row and column swaps of the matrix $\mathcal{Q}$ that will reduce it to block-diagonal form.

Costeira and Kanade have proposed several methods for finding the correct swaps in the presence of noise: one possibility is to minimize the sum of the squares of the off-diagonal block entries over all rows and column permutations (see [?] for details). Figure 13.6 shows experimental results, including the images of two objects and the corresponding feature tracks, a plot of the corresponding shape interaction matrix before and after sorting, and the corresponding segmentation results.

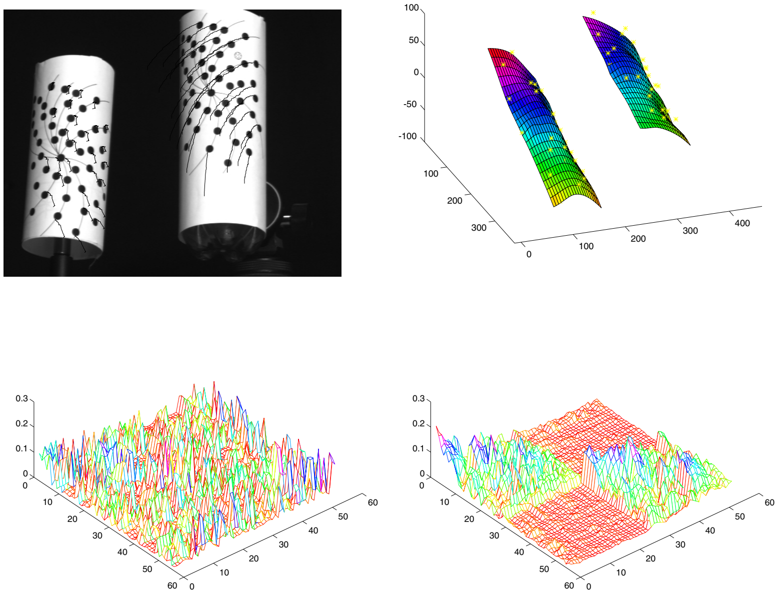


Figure 13.6. Motion segmentation – experimental results: top-left: one frame from a sequence of pictures of two cylinders, including feature tracks; top-right: the recovered shapes after motion segmentation; bottom-left: the shape interaction matrix; bottom-right: the matrix after sorting. Reprinted from [?], Figures 13–15.

13.6 Notes

As shown in this chapter, the stratification of structure from motion into an affine and a Euclidean stage affords simple and robust methods for shape reconstruction from image sequences. The affine stage by itself also affords simple techniques for motion-based image segmentation. As shown in Chapter 25, other applications include interactive image synthesis in the augmented reality domain.

Variations of the rank-3 property of the data matrix associated with an affine motion sequence include the fact that an affine image is the linear combination of three model images [**?**], and that the image trajectories of a scene point are linear combinations of the trajectories of three reference points [**?**].

Various extensions of the approach presented in this chapter have been proposed recently, including the incremental recovery of structure and motion [**?**; **?**], the extension of the affine/metric stratification to a projective/affine/metric one [**?**], along with corresponding projective shape estimation algorithms [**?**; **?**], and the generalization of the factorization approach of Tomasi and Kanade [**?**] to various other computer vision problems that have a natural bilinear structure [**?**].

13.7 Assignments

Exercises

1. In this exercise we prove Theorem 4. Let us define

$$a(\boldsymbol{v}) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{v}\cdot\boldsymbol{a}_i)^2,$$

$\mathcal{S} = (\boldsymbol{a}_1, \dots, \boldsymbol{a}_n)$, and $\mathcal{C} = \frac{1}{n}\mathcal{S}\mathcal{S}^T$. With this notation we have

$$a(\boldsymbol{v}) = \boldsymbol{v}^T\mathcal{C}\boldsymbol{v}.$$

You can assume for simplicity that the eigenvalues of $\mathcal{C}$ are all distinct. Use the following steps to prove Theorem 4.

(a) Show that determining V_p reduces to constructing the orthonormal family of vectors $\boldsymbol{v}_i$ $(i = 1, \dots, p)$ that maximizes $A \overset{\text{def}}{=} \sum_{i=1}^{p} a(\boldsymbol{v}_i)$.

(b) Let $\boldsymbol{u}_j$ $(j = 1, \dots, m)$ denote the eigenvectors of $\mathcal{C}$ with associated eigenvalues λ_i, and let $\boldsymbol{\xi}_i = (\xi_{i1}, \dots, \xi_{im})^T$ denote the coordinate vector of $\boldsymbol{v}_i$ $(i = 1, \dots, p)$ in the basis of $\mathbb{R}^m$ formed by the vectors $\boldsymbol{u}_j$. Show that $A = \sum_{i=1}^{m} \boldsymbol{\xi}_i^T \mathcal{L} \boldsymbol{\xi}_i$, where $\mathcal{L} = \text{Diag}(\lambda_1, \dots, \lambda_m)$.

(c) Let X_p denote the subspace of $\mathbb{R}^m$ spanned by the vectors $\boldsymbol{\xi}_i$ $(i = 1, \dots, p)$ and L_p denote the restriction to X_p of the linear operator $L : \mathbb{R}^m \to \mathbb{R}^m$ associated with the matrix $\mathcal{L}$. Use Lagrange multipliers to show that L_p is an endomorphism of X_p.

(d) Show that X_p is the subspace of $\mathbb{R}^m$ spanned by p of the canonical basis vectors $(1, \dots, 0)$,$\dots$,$(0, \dots, 1)$. (Hint: first show that the matrix representing L_p in the basis of X_p formed by the vectors $\boldsymbol{\xi}_i$ is diagonalizable.)

(e) Show that $\sum_{k=1}^{p} \xi_{ik}^2 = \sum_{k=1}^{p} \xi_{kj}^2 = 1$ $(i, j = 1, \ldots, p)$, and prove the theorem.

2. Show that the matrix $\mathcal{M}\mathcal{M}^\dagger$ associates with any vector $\boldsymbol{d}$ in $\mathbb{R}^{2m}$ its projection onto the three-dimensional subspace $V_{\mathcal{M}}$ spanned by the columns of $\mathcal{M}$. (Hint: shows that $\mathcal{M}\mathcal{M}^\dagger \boldsymbol{d}$ is orthogonal to the columns vectors of $\mathcal{M}$.)

Programming Assignments

Note: the assignments below require routines for numerical linear algebra and singular value decomposition. An extensive set of such routines is available in MATLAB as well as in public-domain libraries such as LINPACK and LAPACK that can be downloaded from the Netlib repository (`http://www.netlib.org/`). Data for these assignments will be available in the CD companion to this book.

1. Implement the Tomasi-Kanade approach to affine shape from motion.

Chapter 14

PROJECTIVE STRUCTURE FROM MOTION

This chapter addresses once again the recovery of the three-dimensional structure of a scene from correspondences established by tracking n points in m pictures. This time, however, we will assume a perspective projection model. Given some fixed world coordinate system, we can write

$$z_{ij}\boldsymbol{p}_{ij} = \mathcal{M}_i\boldsymbol{P}_j \quad \text{for} \quad i = 1, \ldots, m \quad \text{and} \quad j = 1, \ldots, n, \tag{14.0.1}$$

where $\boldsymbol{p}_{ij} = (u_{ij}, v_{ij}, 1)^T$ denotes the (homogeneous) coordinate vector of the projection of the point P_j in the image i expressed in the corresponding camera's coordinate system, z_{ij} is the depth of P_j in the same coordinate system, $\mathcal{M}_i$ is the 3×4 projection matrix associated with this camera in the world coordinate system, and $\boldsymbol{P}_j$ is the (homogeneous) coordinate vector of the point P_j in that coordinate system.

We address the problem of reconstructing both the matrices $\mathcal{M}_i$ $(i = 1, \ldots, m)$ and the vectors $\boldsymbol{P}_j$ $(j = 1, \ldots, n)$ from the image correspondences $\boldsymbol{p}_{ij}$. This problem shares a great deal of similarities with the affine structure-from-motion problem, but it also differs from it in several key aspects: contrary to the set of m affine images of a fixed scene, the set of m perspective images does not exhibit a natural affine structure, or for that matter, any (obvious) simple structure at all. On the other hand, there is a natural ambiguity in perspective structure from motion that is similar (but not identical) to the natural ambiguity of affine structure from motion: in particular, if the camera calibration parameters are unknown, the projection matrices $\mathcal{M}_i$ are, according to Chapter 5, arbitrary 3×4 matrices, and it follows that if $\mathcal{M}_i$ and $\boldsymbol{P}_j$ are solutions of (14.0.1), so are $\mathcal{M}_i\mathcal{Q}$ and $\mathcal{Q}^{-1}\boldsymbol{P}_j$ for any non-singular 4×4 matrix $\mathcal{Q}$ [**?**; **?**].

Linear geometric relations between homogeneous vectors that are invariant under *projective transformations* (i.e. bijective linear mappings associated with 4×4 matrices) fall in the domain of *projective geometry*, which will play in the rest of this chapter the role that affine geometry played in Chapter 13, and will afford a similar overall methodology: once again, ignoring (at first) the Euclidean constraints associated with calibrated cameras will *linearize* the recovery of scene structure and camera motion from point correspondences, and this will allow us to decompose motion analysis into two stages: the first one will be concerned with the recovery of the scene structure and camera motion, up to an arbitrary projective transformation. The second step will exploit the geometric constraints associated with real cameras to upgrade the projective reconstruction to a Euclidean one.

We start by introducing some elementary notions from projective geometry before presenting several algorithms for projective and metric scene and motion reconstruction from point correspondences. The first class of methods presented in this chapter was originally introduced by Faugeras [**?**] and Hartley *et al.* [**?**] in the context of uncalibrated stereo vision, and it takes advantage of the multilinear constraints introduced in Chapter 11 to estimate the scene and motion parameters from a few pictures. We also discuss a different class of techniques that rely on non-linear optimization to exploit the wealth of information contained in long image sequences in a uniform manner [**?**]. We conclude with a discussion of techniques for constructing the metric upgrade of a projective scene reconstruction [**?**; **?**; **?**; **?**].

14.1 Elements of Projective Geometry

Let us consider a real vector space E of dimension $n+1$. If $\boldsymbol{v}$ is a non-zero element of E, the set $\mathbb{R}\boldsymbol{v}$ of all vectors $k\boldsymbol{v}$ as k varies over $\mathbb{R}$ is called a *ray*, and it is uniquely characterized by any one of its non-zero vectors. The *real projective space* $P(E)$ *of dimension* n *associated with* E is the set of rays in E, or equivalently the quotient of the set $E\backslash 0$ of non-zero elements of E under the equivalence relation "$\boldsymbol{u} \sim \boldsymbol{v}$ if and only if $\boldsymbol{u} = k\boldsymbol{v}$ for some $k \in \mathbb{R}$". Elements of $P(E)$ are called points, and we will say that a family of points are linearly dependent (resp. independent) when representative vectors for the corresponding rays are linearly dependent (resp. independent). The map $p : E\backslash 0 \rightarrow P(E)$ associates with any element $\boldsymbol{v}$ of E the corresponding point $p(\boldsymbol{v})$ of $P(E)$.

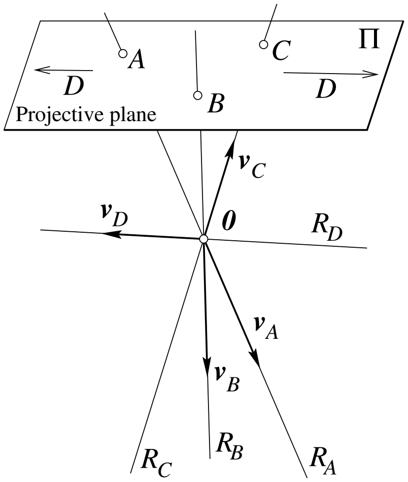
14.1.1 Projective Bases and Projective Coordinates

Consider a basis $(\boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ for E. We can associate with each point A in $P(E)$ a one-parameter family of elements of $\mathbb{R}^{n+1}$, namely the coordinate vectors $(x_0, x_1, \ldots, x_n)^T$ of the vectors $\boldsymbol{v} \in E$ such that $A = p(\boldsymbol{v})$. These tuples are proportional to one another, and a representative tuple is called a set of *homogeneous (projective) coordinates* of the point A.

Homogeneous coordinates can also be characterized intrinsically in terms of families of points in $P(E)$: consider $m+1$ ($m \leq n$) linearly independent points A_i ($i = 0, \ldots, m$) and $m+1$ vectors $\boldsymbol{u}_i$ representative of the corresponding rays. If an additional point A linearly depends upon the points A_i, and $\boldsymbol{u}$ is a representative vector of the corresponding ray, we can write:

$$\boldsymbol{u} = \mu_0 \boldsymbol{u}_0 + \mu_1 \boldsymbol{u}_1 + \ldots + \mu_m \boldsymbol{u}_m.$$

Consider an affine plane Π of $\mathbb{R}^3$. The rays of $\mathbb{R}^3$ that are not parallel to Π are in one-to-one correspondence with the points of this plane. For example, the rays R_A, R_B and R_C associated with the vectors $\boldsymbol{v}_A$, $\boldsymbol{v}_B$ and $\boldsymbol{v}_C$ below can be mapped onto the points A, B and C where they intersect Π. The vectors $\boldsymbol{v}_A$, $\boldsymbol{v}_B$ and $\boldsymbol{v}_C$ are only defined up to scale; they are linearly independent, and so are (by definition) the corresponding points A, B and C.



A model of the projective plane $\mathbb{P}^2 \overset{\text{def}}{=} P(\mathbb{R}^2)$ can be constructed by adding to Π a one-dimensional set of *points at infinity* associated with the rays parallel to this plane. For example, the ray R_D parallel to Π maps onto the point at infinity D. The introduction of points at infinity frees projective geometry from the numerous exceptions encountered in the affine case: for example, parallel affine lines do not intersect unless they coincide. In contrast, any two distinct lines in $\mathbb{P}^2$ intersect in one point, with pairs of parallel lines intersecting at the point at infinity associated with their common direction.

Example 14.1: *A projective plane embedded in* $\mathbb{R}^3$.

Note that the coefficients μ_i are not uniquely determined since *each* vector $\boldsymbol{u}_i$ is only defined up to a non-zero scale factor. However, when none of the coefficients μ_i vanishes, i.e., when $\boldsymbol{u}$ does not lie in the vector subspace spanned by any m vectors $\boldsymbol{u}_i$, we can uniquely define the $m+1$ non-zero vectors $\boldsymbol{e}_i = \mu_i \boldsymbol{u}_i$ such that

$$\boldsymbol{u} = \boldsymbol{e}_0 + \boldsymbol{e}_1 + \ldots + \boldsymbol{e}_m.$$

Any other vector $\boldsymbol{v}$ linearly dependent on the vectors $\boldsymbol{u}_i$ can now be written *uniquely* as

$$\boldsymbol{v} = x_0 \boldsymbol{e}_0 + x_1 \boldsymbol{e}_1 + \ldots + x_m \boldsymbol{e}_m.$$

This defines a one-to-one correspondence between the rays $\mathbb{R}(x_0, x_1, \ldots, x_m)^T$ of $\mathbb{R}^{m+1}$ and a linear subspace S_m of $P(E)$. S_m is in fact the projective space $P(E_m)$ associated with the vector subspace E_m of E spanned by the vectors $\boldsymbol{u}_i$ (or equivalently of course by the vectors $\boldsymbol{e}_i$). The number m is called the dimension of S_m. If $P = p(\boldsymbol{v})$ is the point of S_m associated with the ray $\mathbb{R}\boldsymbol{v}$, the numbers $x_0, x_1, \ldots, x_m$ are called the homogeneous (projective) coordinates of P in the *projective basis* (or *projective frame*) determined by the $m+1$ *fundamental points* A_i and the *unit point* A. Note that, since the vector $\boldsymbol{v}$ associated with a ray is only defined up to scale, so are the homogeneous coordinates of a point.

The coordinate vectors of the points that form a projective basis have a particularly simple form: in particular, since the vectors $\boldsymbol{u}_i$ (hence the vectors $\boldsymbol{e}_i$) are linearly independent, the coordinate vectors of the fundamental points A_i $(i = 0, \ldots, m)$ are

$$\begin{cases} \boldsymbol{A}_0 &= (1, 0, \ldots, 0)^T, \\ \boldsymbol{A}_1 &= (0, 1, \ldots, 0)^T, \\ \ldots & \\ \boldsymbol{A}_m &= (0, 0, \ldots, 1)^T. \end{cases}$$

The coordinate vector of the unit point A is, by definition, $\boldsymbol{A} = (1, 1, \ldots, 1)^T$.

Example 14.2: *Coordinate vectors of a projective basis.*

It should be clear that the two notions of homogeneous coordinates that have been introduced in this section coincide. The only difference is in the choice of the coordinate vectors $\boldsymbol{e}_i$ $(i = 0, \ldots, m)$, that are given a priori in the former case, and constructed from the points forming a given projective frame in the latter one.

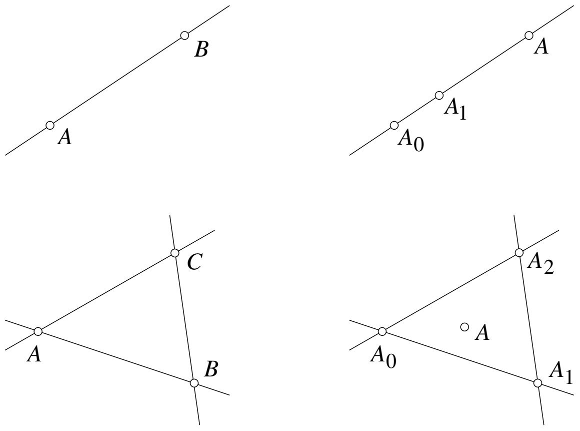
A linear subspace S_1 of dimension 1 of $\mathbb{P}^n$ is called a *line*. Linear subspaces of dimension 2 and $n-1$ are respectively called *planes* and *hyperplanes*. A hyperplane S_{n-1} consists of the set of points P linearly dependent on n linearly independent points P_i $(i = 1, \ldots, n)$.

14.1.2 Projective Transformations

Consider an injective linear map $U : E \to F$ between two vector spaces E and F. By linearity, U maps rays of E onto rays of F. Since it is injective, it also maps non-zero vectors onto non-zero vectors and therefore induces a map $P(U) : P(E) \to P(F)$ between the quotient spaces $P(E)$ and $P(F)$. This map is called a *projective map*, and a *projective transformation* (or *homography*) if it is bijective. It is easy to show that projective transformations form a group under the law of composition of maps. This group is called the *projective group* of $P(E)$.

Now consider two n-dimensional projective spaces $P(E)$ and $P(E')$, equipped respectively with the coordinate frames $(A_0, A_1, \ldots, A_{n+1})$ and $(A'_0, A'_1, \ldots, A'_{n+1})$ (here A_{n+1} and A'_{n+1} are the unit points of the two frames). There exists a unique

A line is uniquely determined by two distinct points lying on it, but a projective frame for a line is determined by *three* distinct points on that line.



Likewise, a plane is uniquely determined by three points lying in it, but a projective frame for that plane is defined by four points: three fundamental points forming a non-degenerate triangle and a unit point not lying on one of the edges of this triangle.

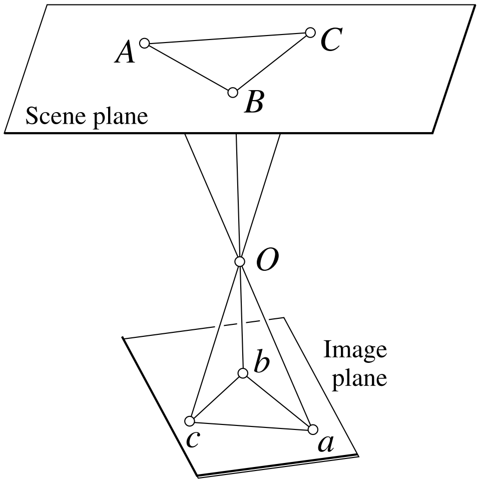
Example 14.3: *Lines and planes.*

homography $U : P(E) \rightarrow P(E')$ such that $U(A_i) = A_i'$ for $i = 0, \ldots, n+1$. This is often refered to as the *first fundamental theorem of projective geometry.*

Given some choice of coordinate frame, projective transformations can conveniently be represented by matrices. Let $\boldsymbol{P}$ denote the coordinate vector of the point P in $P(E)$ and $\boldsymbol{P}'$ denote the coordinate vector of the point P' in $P(E')$; if $\mathcal{U}$ is a non-singular $(n+1) \times (n+1)$ matrix, the equation $\boldsymbol{P}' = \mathcal{U}\boldsymbol{P}$ defines a homography between the points P of $P(E)$ and the points P' of $P(E')$. Conversely, any projective transformation $U : P(E) \rightarrow P(E')$ can also be represented by a non-singular $(n+1) \times (n+1)$ matrix $\mathcal{U}$.

Projective geometry can be thought of as the study of the properties of a projective space $P(E)$ that are invariant under any non-singular projectivity. An example of such a property is the linear dependence (or independence) of a family of points in $P(E)$. Another fundamental example is obtained by considering a basis $(A_0, A_1, \ldots, A_{n+1})$ of $P(E)$ and constructing the images $A_i' = U(A_i)$ of its points via the projective transformation $U : P(E) \rightarrow P(E')$. The points A_i' form a basis for $P(E')$ and if a point P has coordinates $(x_0, x_1, \ldots, x_n)$ in the basis $(A_0, A_1, \ldots, A_{n+1})$, the point $U(P)$ has the same coordinates in the basis $(A_0', A_1', \ldots, A_{n+1}')$. Coming back to Example 14.4, it follows that an image of a set of coplanar points completely determines the projective coordinates of these

Consider two planes and a point O in $\mathbb{P}^3$. As shown in the exercises, the perspective projection mapping any point A in the first plane onto the intersection of the line AO and the second plane is a projective transformation.



Example 14.4: *Projective correspondence between coplanar points and their pictures.*

points relative to the frame formed by four of them. This will prove very useful in designing invariant-based recognition systems in latter chapters.

Given two projective bases $(A'_0, A'_1, \dots, A_{n+1})$ and $(A''_0, A''_1, \dots, A_n + 1)$ of the n-dimensional projective space $P(E)$, it is also easily shown (see exercises) that the coordinate vectors $\boldsymbol{P}'$ and $\boldsymbol{P}''$ of the same point P are related by $\boldsymbol{P}'' = \mathcal{A}\boldsymbol{P}'$, where $\mathcal{A}$ is as before an $(n+1) \times (n+1)$ non-singular matrix.

14.1.3 Affine and Projective Spaces

Example 14.1 introduced (informally) the idea of embedding an affine plane into a projective one with the addition of a line at infinity. More generally, it is possible to construct the *projective closure* $\hat{X}$ of an affine space X of dimension n by adding to it a set of points at infinity associated with the directions of its lines. These points form a hyperplane of $\hat{X}$, called the *hyperplane at infinity* and denoted by ∞_X.

Let us pick some point A in X and introduce $\hat{X} \overset{\text{def}}{=} P(\vec{X} \times \mathbb{R})$. We can embed X into $\hat{X}$ via the injective map $J_A : X \to \hat{X}$ defined by $J_A(P) = p(\overrightarrow{AP}, 1)$.[1]

---

[1] Here we identify X and the underlying vector space $\vec{X}$ by identifying each point P in X with the vector $\overrightarrow{AP}$. This *vectorialization* process is of course dependent on the choice of the origin A, but it can easily be shown that $\hat{X}$ is indeed independent of that choice. A more rigorous approach to the projective completion process involves the *universal vector space* associated with an affine

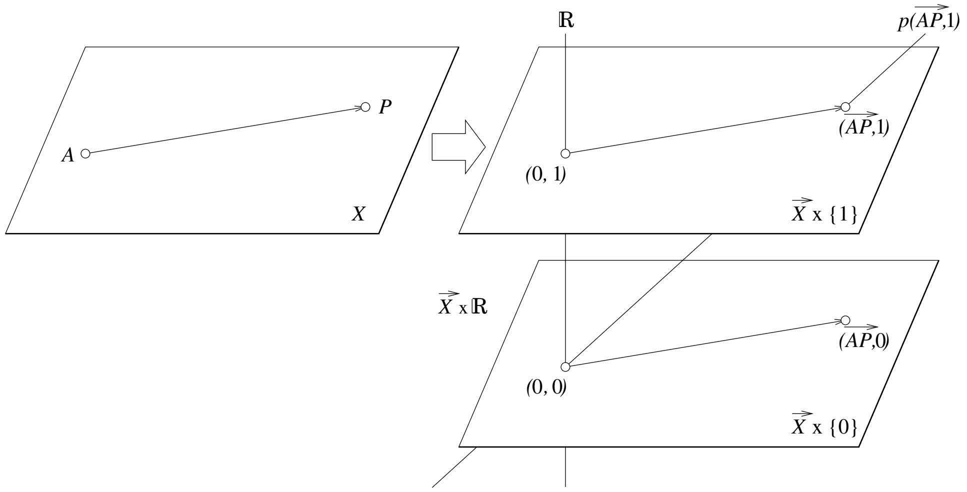


Figure 14.1. The projective completion of an affine space.

The complement of X in $\hat{X}$ is the hyperplane at infinity $\infty_X \overset{\text{def}}{=} P(\vec{X} \times \{0\}) \approx P(\vec{X})$ mentioned earlier. We can also relate affine and projective coordinates as follows. Consider a fixed affine frame $(A_0, A_1, \ldots, A_n)$ of X and embed X into $\hat{X}$ using J_{A_0}. The vectors $\boldsymbol{e}_i \overset{\text{def}}{=} \overrightarrow{A_0 A_i}$ $(i = 1, \ldots, n)$ form a basis for $\vec{X}$, thus the vectors $(\boldsymbol{e}_1, 0)$, …, $(\boldsymbol{e}_n, 0)$, and $(0, 1)$ form a basis of $\vec{X} \times \mathbb{R}$. If P has coordinates $(x_1, \ldots, x_n)$ in the basis formed by the points A_i, then $J_{A_0}(P)$ has coordinates $(x_1, \ldots, x_n, 1)$ in the basis formed by these $n + 1$ vectors (the elements of ∞_X, on the other hand, have coordinates of the form $(x_1, \ldots, x_n, 0)$). Note that the projective completion process justifies, at long last, the representation of image and scene points by homogeneous coordinates introduced in Chapter 5 and used throughout this book.

14.1.4 Hyperplanes and Duality

As mentioned before, the introduction of hyperplanes at infinity frees projective geometry from the exceptions that plague affine geometry. For example, in the projective plane, two distinct lines have exactly one common point (possibly at infinity). Likewise, two distinct points belong to exactly one line. These two statements can actually be taken as *incidence axioms*, from which the projective plane can be constructed axiomatically. Points and lines play a symmetric, or more precisely *dual* role in these statements.

To introduce duality a bit more generally, let us equip the n-dimensional projective space $P(E)$ with a fixed projective frame and consider $n+1$ points $P_0, P_1, \ldots, P_n$

---

space and would be out of place here. See [?, Chapter 5] for details.

lying in some hyperplane S_{n-1} of $P(E)$. Since these points are by construction linearly dependent, the $(n+1) \times (n+1)$ matrix formed by collecting their coordinate vectors is singular. Expanding its determinant with respect to its last column yields

$$u_0 x_0 + u_1 x_1 + \ldots + u_n x_n = 0, \tag{14.1.1}$$

where $(x_0, x_1, \ldots, x_n)$ denote the coordinates of P_n and the coefficients u_i $(i = 0, 1, \ldots, n)$ are functions of the coordinates of the points P_j $(j = 0, 1, \ldots, n-1)$.

Equation (14.1.1) is satisfied by every point P_n in the hyperplane S_{n-1}, and it is called the equation of S_{n-1}. Conversely, it is easily shown (see exercises) that any equation of the form (14.1.1) where at least one of the coefficients u_i is non-zero is the equation of some hyperplane. Since the coefficients u_i in (14.1.1) are only defined up to some common scale factor, there exists a one-to-one correspondence between the rays of $\mathbb{R}^{n+1}$ and the hyperplanes of $P(E)$, and it follows that we can define a second projective space $P(E^*)$ formed by these hyperplanes (this notation is justified by the fact that $P(E^*)$ can be shown to be the projective space associated with the dual vector space E^* of E).

It can also be shown that any geometric theorem that holds for points in $P(E)$ induces a corresponding theorem for hyperplanes (i.e., points in $P(E^*)$) and vice versa. The two theorems are said to be dual of each other. For example, points and lines are dual notions in $\mathbb{P}^2$, while points and planes are dual in $\mathbb{P}^3$.[2]

14.1.5 Cross-Ratios

From now on we will focus on the three-dimensional projective space formed by the projective closure of the physical affine space surrounding us. Non-homogeneous projective coordinates of a point can be defined geometrically in terms of *cross-ratios*: in the affine case, given four collinear points A, B, C, D such that A, B and C are distinct, we define the cross-ratio of these points as

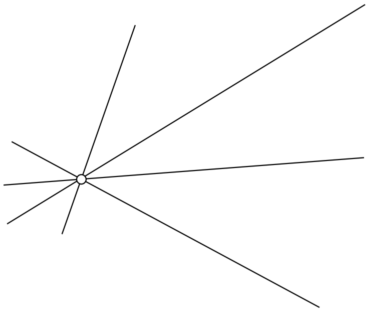
$$h_{ABC}(D) = [A, B, C, D] \overset{\text{def}}{=} \frac{\overline{CA}}{\overline{CB}} \times \frac{\overline{DB}}{\overline{DA}},$$

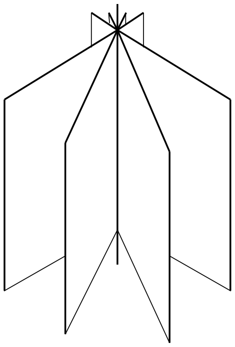
where $\overline{PQ}$ denotes the signed distance between two points P and Q for some choice of orientation of the line Δ joining them. The orientation of this line is fixed but arbitrary, since reversing it will obviously not change the cross-ratio. Note that $[A, B, C, D]$ is, a priori, only defined when $D \neq A$ since its calculation involves a division by zero when $D = A$. We extend the definition of the cross-ratio to the whole affine line by using the symbol ∞ to denote the ratio formed by dividing

---

[2] At this point we cannot resist quoting Samuel [**?**], who mentions an early ninenteeth-century controversy on the basis of duality between points and lines in the plane. After contrasting the views of Gergonne and Poncelet (the former, that emphasizes the structural similarity of the projective spaces formed by points and lines, is closer to the modern view of duality), Samuel goes on to write: "Since Poncelet was a general and the head of the Ecole Polytechnique, and Gergonne a mere captain in the French artillery, it was the former's point of view that prevailed, at least among their French contemporaries."

What is the dual of a line of $P(E)$? A line is a one-dimensional linear subspace of $P(E)$, whose elements are linearly dependent on two points on the line. Likewise, a line of $P(E^*)$ is a one-dimensional subspace of the dual, called a *pencil of hyperplanes*, whose elements are linearly dependent on two hyperplanes in the pencil. In the plane, the dual of a line is a pencil of lines intersecting at a common point.





In three dimensions, the dual of a line is a pencil of planes, intersecting along a common line.

Example 14.5: *Duality*

any non-zero real number by zero, and to the whole projective line $\hat{\Delta}$ by defining $[A, B, C, \infty_\Delta] = \overline{CA}/\overline{CB}$. Alternatively, given three points A, B and C on a *projective* line Δ, the cross-ratio can also be defined as the unique projective transformation $h_{ABC} : \Delta \to \hat{\mathbb{R}}$ mapping Δ onto the projective completion $\hat{\mathbb{R}} = \mathbb{R} \cup \infty$ of the real line such that $h(A) = \infty$, $h(B) = 0$ and $h(C) = 1$.

Given a projective frame (A_0, A_1, A) for a line Δ, and a point P lying on Δ with homogeneous coordinates (x_0, x_1) in that frame, we can define a non-homogeneous coordinate for P as $k_0 = x_0/x_1$. The scalar k_0 is sometimes called *projective parameter* of P, and it is easy to show that $k_0 = [A_0, A_1, A_2, P]$.

As noted earlier, a set of lines passing through the same point O is called a pencil of lines. The cross-ratio of four *coplanar* lines Δ_1, Δ_2, Δ_3 and Δ_4 in some pencil is defined as the cross-ratio of the intersections of these lines with any other line Δ in the same plane that does not pass through O, and it is easily shown to be independent of the choice of Δ (Figure 14.2).

Consider now four planes Π_1, Π_2, Π_3 and Π_4 in the same pencil, and denote by Δ their common line. The cross-ratio of these planes is defined as the cross-ratio of the pencil of lines formed by their intersection with any other plane Π not containing Δ (Figure 14.3). Once again, the cross-ratio is easily shown to be independent of the choice of Π.

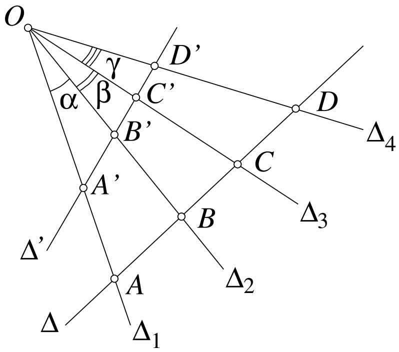


Figure 14.2. Definition of the cross-ratio of four lines. As shown in the exercises, the cross-ratio $[A, B, C, D]$ depends only on the three angles α, β and γ. In particular, we have $[A, B, C, D] = [A', B', C', D']$.

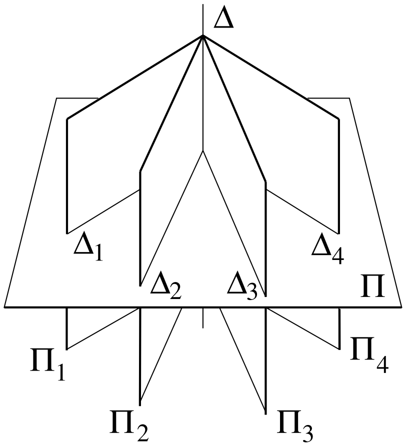


Figure 14.3. The cross-ratio of four planes.

In the plane, the non-homogeneous projective coordinates (k_0, k_1) of the point P in the basis (A_0, A_1, A_2, A) are defined by $k_0 = x_0/x_2$ and $k_1 = x_1/x_2$, and it can be shown that

$$\begin{cases} k_0 = [A_1A_0, A_1A_2, A_1A, A_1P], \\ k_1 = [A_0A_1, A_0A_2, A_0A, A_0P], \end{cases}$$

where MN denotes the line joining the points M and N, and $[\Delta_1, \Delta_2, \Delta_3, \Delta_4]$ denotes the cross-ratio of the pencil of lines Δ_1, Δ_2, Δ_3, Δ_4.

Similarly, the non-homogeneous projective coordinates (k_0, k_1, k_2) of the point P in the basis (A_0, A_1, A_2, A_3, A) are defined by $k_0 = x_0/x_3$, $k_1 = x_1/x_3$ and

$k_2 = x_2/x_3$, and it can be shown that

$$\begin{cases} k_0 = [A_1A_2A_0, A_1A_2A_3, A_1A_2A, A_1A_2P], \\ k_1 = [A_2A_0A_1, A_2A_0A_3, A_2A_0A, A_2A_0P], \\ k_2 = [A_0A_1A_2, A_0A_1A_3, A_0A_1A, A_0A_1P], \end{cases}$$

where LMN denotes the plane spanned by the three points L, M and N, and $[\Pi_1, \Pi_2, \Pi_3, \Pi_4]$ denotes the cross-ratio of the pencil of planes Π_1, Π_2, Π_3, Π_4.

14.1.6 Application: Parameterizing the Fundamental Matrix

Before attacking the main topic of this chapter, i.e., the estimation of the projective structure of a scene from multiple images, let us give as an example an important application of projective geometry to vision by revisiting the problem of determining the epipolar geometry of two uncalibrated cameras. This problem was introduced in Chapter 11, where we gave without proof an explicit parameterization of the fundamentel matrix. We now construct this parameterization. Let us denote by e and e' the epipoles associated with the two images and define the *epipolar transformation* as the mapping from one set of epipolar lines onto the other one: as shown in [**?**] and illustrated by Figure 14.4, this transformation is a homography. Indeed, the epipolar planes associated with the two cameras form a pencil whose spine is the baseline joining the two optical centers. This pencil intersects the corresponding image planes along the two families of epipolar lines, and the cross-ratio of any quadruple of lines in either family is of course the same as the cross-ratio of the corresponding planes. In turn, this means that the epipolar transformation preserves the cross-ratio and is therefore a projective tranformation.

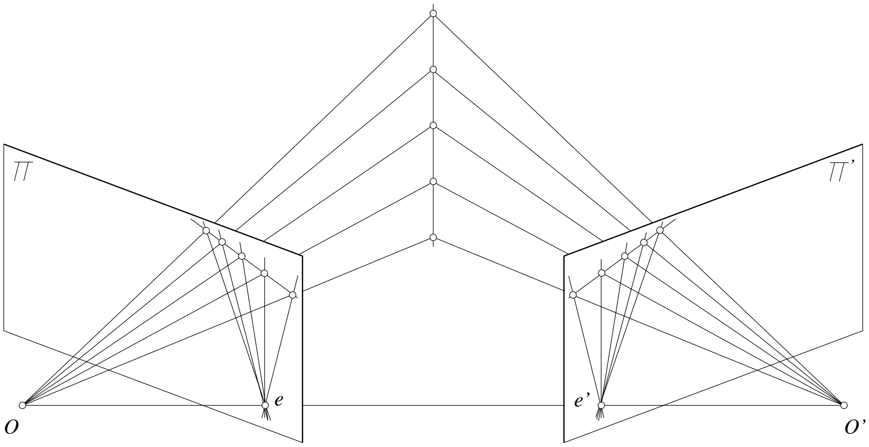


Figure 14.4. Epipolar geometry: corresponding epipolar lines in the two images form pencils of lines in projective correspondence.

In particular, if the epipolar line l with slope τ in the first image is in correspondence with the epipolar line l' with slope τ' in the second image, then, as shown in the exercises,

$$\tau \rightarrow \tau' = \frac{a\tau + b}{c\tau + d}, \tag{14.1.2}$$

where a, b, c, d are the coefficients of the homography, and

$$\tau = \frac{v - \beta}{u - \alpha}, \quad \tau' = \frac{v' - \beta'}{u' - \alpha'},$$

where $\boldsymbol{p} = (u, v)^T$ and $\boldsymbol{p}' = (u', v')^T$ are the coordinate vectors of corresponding points, and $\boldsymbol{e} = (\alpha, \beta)^T$ and $\boldsymbol{e}' = (\alpha', \beta')^T$ are the positions of the epipoles. This homography is the epipolar transformation. As shown in [**?**] and in the exercises, the coefficients of the fundamental matrix can be computed from the positions of the epipoles and a, b, c, d, and vice versa. In particular, we obtain the parameterization of $\mathcal{F}$ given without proof in Chapter 11, i.e.,

$$\mathcal{F} = \begin{pmatrix} b & a & -a\beta - b\alpha \\ -d & -c & c\beta + d\alpha \\ d\beta' - b\alpha' & c\beta' - a\alpha' & -c\beta\beta' - d\beta'\alpha + a\beta\alpha' + b\alpha\alpha' \end{pmatrix}.$$

14.2 Projective Scene Reconstruction from Two Views

The rest of this chapter is concerned with the recovery of the three-dimensional structure of a scene assuming that n points have been tracked in m images of this scene. This section focuses on the case of two images. The general multi-view problem will be revisited in the next two sections.

14.2.1 Analytical Scene Reconstruction

The perspective projection equations introduced in Chapter 5 extend naturally to the projective completion of the physical three-dimensional affine space. In particular, let us consider five points A_i $(i = 0, \ldots, 4)$ and choose them as a basis for this projective space, with A_4 playing the role of the unit point. We consider a camera observing these points, with projection matrix $\mathcal{M}$, and denote by a_i $(i = 0, \ldots, 4)$ the images of these points, and choose the points a_0 to a_3 as a projective basis of the image plane, a_3 being this time the unit point. We also denote by α, β and γ the coordinates of a_4 in this basis.

Writing, for $i = 0, \ldots, 4$ that $z_i \boldsymbol{a}_i = \mathcal{M}\boldsymbol{A}_i$ yields immediately

$$\mathcal{M} = \begin{pmatrix} z_0 & 0 & 0 & z_3 \\ 0 & z_1 & 0 & z_3 \\ 0 & 0 & z_2 & z_3 \end{pmatrix} \quad \text{and} \quad \begin{cases} z_4\alpha = z_0 + z_3, \\ z_4\beta = z_1 + z_3, \\ z_4\gamma = z_2 + z_3. \end{cases}$$

Since a perspective projection matrix is only defined up to scale, we can divide its coefficients by z_3, and defining $\lambda = z_4/z_3$ yields

$$\mathcal{M} = \begin{pmatrix} \lambda\alpha - 1 & 0 & 0 & 1 \\ 0 & \lambda\beta - 1 & 0 & 1 \\ 0 & 0 & \lambda\gamma - 1 & 1 \end{pmatrix}.$$

Let us now suppose we have a second image of the same scene, with projection matrix $\mathcal{M}'$ and image points a'_i ($i = 0, \dots, 4$). The same construction applies in this case, and we obtain

$$\mathcal{M}' = \begin{pmatrix} \lambda'\alpha' - 1 & 0 & 0 & 1 \\ 0 & \lambda'\beta' - 1 & 0 & 1 \\ 0 & 0 & \lambda'\gamma' - 1 & 1 \end{pmatrix}.$$

The stereo configuration of our two cameras is thus completely determined by the two parameters λ and λ'. The epipolar geometry of the rig can now be used to compute these parameters: let us denote by C the optical center of the first camera and by e' the associated epipole in the image plane of the second camera, with coordinate vectors $\boldsymbol{C}$ and $\boldsymbol{e}'$ in the corresponding projective bases. We have $\mathcal{M}\boldsymbol{C} = \boldsymbol{0}$ thus

$$\boldsymbol{C} = (\frac{1}{1-\lambda\alpha}, \frac{1}{1-\lambda\beta}, \frac{1}{1-\lambda\gamma}, 1)^T,$$

and substituting in the equation $\mathcal{M}'\boldsymbol{C} = \boldsymbol{e}'$ yields

$$\boldsymbol{e}' = (1 - \frac{\lambda'\alpha' - 1}{\lambda\alpha - 1}, 1 - \frac{\lambda'\beta' - 1}{\lambda\beta - 1}, 1 - \frac{\lambda'\gamma' - 1}{\lambda\gamma - 1})^T$$

Now, if μ' and ν' denote this time the *non-homogeneous* coordinates of e' in the projective basis formed by the points a'_i, we finally obtain

$$\begin{cases} \mu'(\lambda\gamma - \lambda'\gamma')(\lambda\alpha - 1) = (\lambda\alpha - \lambda'\alpha')(\lambda\gamma - 1), \\ \nu'(\lambda\gamma - \lambda'\gamma')(\lambda\beta - 1) = (\lambda\beta - \lambda'\beta')(\lambda\gamma - 1). \end{cases} \tag{14.2.1}$$

A system of two quadratic equations in two unknowns λ and λ' such as (14.2.1) admits in general four solutions, that can be thought of as the four intersections of the conic sections defined by the two equations in the (λ, λ') plane. Inspection of (14.2.1) reveals immediately that $(\lambda, \lambda') = (0, 0)$ and $(\lambda, \lambda') = (1/\gamma, 1/\gamma')$ are always solutions of these equations. It is easy (if a bit tedious) to show that the two remaining solutions are identical (geometrically the two conics are tangent to each other at their point of intersection), and that the corresponding values of the parameters λ and λ' are given by

$$\lambda = \frac{\operatorname{Det}\begin{pmatrix} \mu' & \alpha & \alpha' \\ \nu' & \beta & \beta' \\ 1 & \gamma & \gamma' \end{pmatrix}}{\operatorname{Det}\begin{pmatrix} \mu'\alpha & \alpha & \alpha' \\ \nu'\beta & \beta & \beta' \\ \gamma & \gamma & \gamma' \end{pmatrix}} \quad \text{and} \quad \lambda' = \frac{\operatorname{Det}\begin{pmatrix} \mu & \alpha & \alpha' \\ \nu & \beta & \beta' \\ 1 & \gamma & \gamma' \end{pmatrix}}{\operatorname{Det}\begin{pmatrix} \mu\alpha' & \alpha & \alpha' \\ \nu\beta' & \beta & \beta' \\ \gamma' & \gamma & \gamma' \end{pmatrix}}.$$

These values uniquely determine the projection matrices $\mathcal{M}$ and $\mathcal{M}'$. Note that taking into account the equations defining the second epipole would not add independent constraints because of the epipolar constraint that relates matching epipolar lines. Once the projection matrices are known, it is a simple matter to reconstruct the scene points.

14.2.2 Geometric Scene Reconstruction

We now give a simple geometric alternative to the analytical approach presened in the previous section. Choosing the optical centers of the cameras as part of the points that define the projective frame will simplify the construction in this case.

Let us start by fixing the notation (Figure 14.5). Suppose that we observe four non-coplanar points A, B, C, D with a weakly calibrated stereo rig. Let O' (resp. O'') denote the position of the optical center of the first (resp. second) camera. For any point P, let p' (resp. p'') denote the position of the projection of P into the first (resp. second) image, and P' (resp. P'') denote the intersection of the ray $O'P$ (resp. $O''P$) with the plane ABC. The epipoles are e' and e'' and the baseline intersects the plane ABC in E. (Clearly, $E' = E'' = E$, $A' = A'' = A$, etc.)

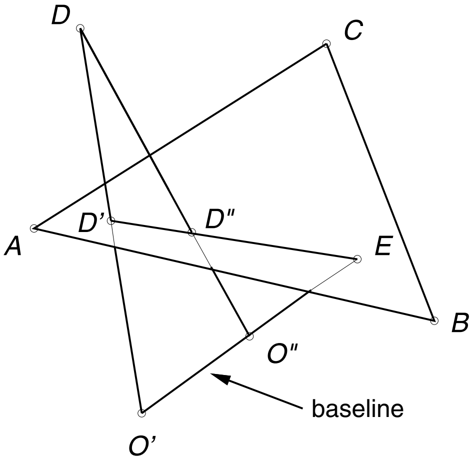


Figure 14.5. Geometry of the three-point problem.

We choose A, B, C, O', O'' as a basis for projective three-space, and our goal is to reconstruct the position of D. Choosing a', b', c', e' as a basis for the first image plane, we can measure the coordinates of d' in this basis and reconstruct the point D' in the basis A, B, C, E of the plane ABC. Similarly, we can reconstruct the point D'' from the projective coordinates of d'' in the basis a'', b'', c'', e'' of the second image plane. The point D is finally reconstructed as the intersection of the two lines $O'D'$ and $O''D''$.

We can now express this geometric construction in algebraic terms. It turns out to be simpler to reorder the points of our projective frame and to calculate the non-homogeneous projective coordinates of D in the basis formed by the tetrahedron A, O'', O', B and the unit point C. These coordinates are defined by the following three cross-ratios:

$$\begin{cases} k_0 = [O''O'A, O''O'B, O''O'C, O''O'D]\,, \\ k_1 = [O'AO'', O'AB, O'AC, O'AD]\,, \\ k_2 = [AO''O', AO''B, AO''C, AO''D]\,. \end{cases} \tag{14.2.2}$$

By intersecting the corresponding pencils of planes with the two image planes we immediately obtain the values of k_0, k_1, k_2 as cross-ratios directly measurable in the two images:

$$\begin{cases} k_0 = [e'a', e'b', e'c', e'd'] = [e''a'', e''b'', e''c'', e''d'']\,, \\ k_1 = [a'e', a'b', a'c', a'd']\,, \\ k_2 = [a''e'', a''b'', a''c'', a''d'']\,. \end{cases} \tag{14.2.3}$$

Note that, for any choice of positions for the reference points, k_0, k_1, and k_2 can be used to reconstruct D as the intersection of the three planes $O'O''D$, $AO'D$, and $AO''D$. As shown in the exercises, similar methods can be used to reconstruct the scene when the five basis points are chosen arbitrarily.

Figure 14.6 illustrates this method with data consisting of 46 point correspondences established between two images taken by weakly-calibrated cameras. Figure 14.6(a) shows the input images and point matches. Figure 14.6(b) shows a view of the corresponding projective scene reconstruction, the raw projectives coordinates being used for rendering purposes. Since this form of display is not particularly enlightening, we also show in Figure 14.6(c) the reconstruction obtained by applying to the scene points the projective transformation mapping the three reference points (shown as small circles) onto their calibrated Euclidean positions. The true point positions are displayed as well for comparison.

14.3 Motion Estimation from Two or Three Views

The methods given in the previous two sections reconstruct the scene relative to five of its points, thus the quality of the reconstruction will strongly depend on the accuracy of the localization of these points in the two images. In contrast, the approach presented in this section takes all points into account in a uniform manner and uses the multilinear constraints introduced in Chapter 11 to reconstruct the camera motion in the form of the associated projection matrices.

14.3.1 Motion Estimation from Fundamental Matrices

As seen in Chapter 11, the general form of the fundamental matrix is

$$\mathcal{F} = \mathcal{K}^{-T}[\boldsymbol{t}_\times]\mathcal{R}\mathcal{K}'^{-1},$$

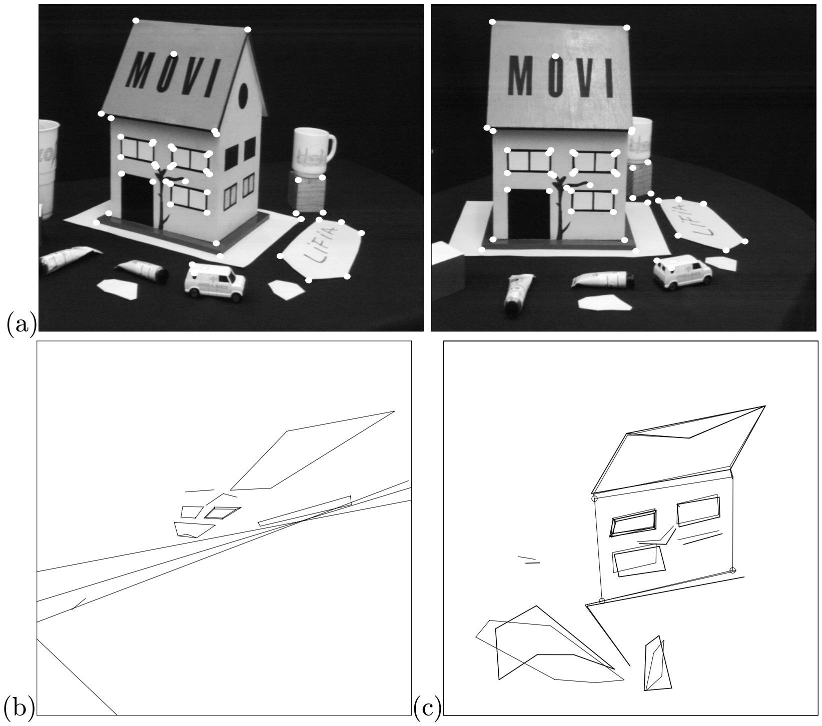


Figure 14.6. Geometric point reconstruction: (a) input data; (b) raw projective coordinates; (c) corrected projective coordinates. Reprinted from [?], Figures 1 and 9.

where the projection matrices associated with the two cameras are $\mathcal{M} = (\,\mathcal{K} \quad \mathbf{0}\,)$ and $\mathcal{M}' = \mathcal{K}'\mathcal{R}^T\,(\,\mathrm{Id} \quad -\boldsymbol{t}\,)$. Here, $\mathcal{K}$ and $\mathcal{K}'$ are 3×3 calibration matrices, and $\mathcal{R}$ and $\boldsymbol{t}$ specify the rigid transformation relating the two cameras' coordinate systems.

Since in the projective setting the scene structure and the camera motion are only defined up to an arbitrary projective transformation, we can always reduce the first projection matrix to the canonical form $\mathcal{M} = (\,\mathrm{Id} \quad 0\,)$ by postmultiplying it by the matrix

$$\mathcal{H} = \begin{pmatrix} \mathcal{K}^{-1} & \mathbf{0} \\ \boldsymbol{\alpha}^T & \beta \end{pmatrix},$$

where $\boldsymbol{\alpha}$ is an arbitrary element of $\mathbb{R}^3$ and β is an arbitrary non-zero real number. It is easy to show that this four-parameter family of matrices is the most general class of projective transformations that achieve the desired reduction.

Let us define $\mathcal{A} = \mathcal{K}'\mathcal{R}^T\mathcal{K}^{-1}$ and $\boldsymbol{b} = -\beta\mathcal{K}'\mathcal{R}^T\boldsymbol{t}$. When postmultiplied by $\mathcal{H}$, the second projection matrix takes the general form

$$\mathcal{M}' = (\,\mathcal{B} \quad \boldsymbol{b}\,), \quad \text{where} \quad \mathcal{B} = \mathcal{A} + \frac{1}{\beta}\boldsymbol{b}\boldsymbol{\alpha}^T. \tag{14.3.1}$$

The vector $\boldsymbol{b}$ can be thought of as the homogeneous coordinate vector of the second epipole in the corresponding image coordinate system.

Let us derive an alternate expression for the fundamental matrix using the new form of the projection matrices. If $\boldsymbol{P} \in \mathbb{R}^3$ denotes the *non-homogeneous* coordinate vector of the point P in the world coordinate system, we can write the projection equations associated with the two cameras as

$$z\boldsymbol{p} = \boldsymbol{P} \quad \text{and} \quad z'\boldsymbol{p}' = \mathcal{B}\boldsymbol{P} + \boldsymbol{b},$$

which can be rewritten as

$$z'\boldsymbol{p}' = z\mathcal{B}\boldsymbol{p} + \boldsymbol{b}.$$

It follows that $z'\boldsymbol{b}\times\boldsymbol{p}' = z\boldsymbol{b}\times\mathcal{B}\boldsymbol{p}$, and forming the dot product of this expression with $\boldsymbol{p}'$ finally yields

$$\boldsymbol{p}^T\mathcal{F}\boldsymbol{p}' = 0 \quad \text{where} \quad \mathcal{F} \overset{\text{def}}{=} \mathcal{B}^T[\boldsymbol{b}_\times].$$

This parameterization of the matrix $\mathcal{F}$ provides a simple method for computing the projection matrix $\mathcal{M}'$. First note that since the overall scale of $\mathcal{M}'$ is irrelevant, we can always take $|\boldsymbol{b}| = 1$. Under this constraint, $\mathcal{M}'$ is still only defined up to the four-parameter class of transformations specified by (14.3.1), and this allows us to choose $\boldsymbol{\alpha} = -\beta\mathcal{A}^T\boldsymbol{b}$, which in turn ensures that $\mathcal{B}^T\boldsymbol{b} = 0$ when $\boldsymbol{b}$ has unit norm. These choices allow us to first compute $\boldsymbol{b}$ as the linear least-squares solution of $\mathcal{F}\boldsymbol{b} = 0$ with unit norm, then pick $\mathcal{B} = [\boldsymbol{b}_\times]\mathcal{F}^T$ since

$$[\boldsymbol{b}_\times]\mathcal{F}^T = -[\boldsymbol{b}_\times]^2\mathcal{B}$$

and it is easy to show for any vector $\boldsymbol{a}$, $[\boldsymbol{a}_\times]^2 = \boldsymbol{a}\boldsymbol{a}^T - |\boldsymbol{a}|^2\text{Id}$.

Once the matrix $\mathcal{M}$ is known, we can compute the position of any point P by solving in the least-squares sense the non-homogeneous linear system of equations in z and z' defined by $z'\boldsymbol{p}' = z\mathcal{B}\boldsymbol{p} + \boldsymbol{b}$. Various alternatives to this technique are discussed in [**?**; **?**].

14.3.2 Motion Estimation from Trifocal Tensors

Here we rewrite the uncalibrated trilinear constraints derived in Chapter 11 in a projective setting. Recall that we wrote the projection matrices associated with three cameras as

$$\mathcal{M}_1 = (\,\mathcal{K}_1 \quad \boldsymbol{0}\,), \quad \mathcal{M}_2 = (\,\mathcal{A}_2\mathcal{K}_1 \quad \boldsymbol{a}_2\,) \quad \text{and} \quad \mathcal{M}_3 = (\,\mathcal{A}_3\mathcal{K}_1 \quad \boldsymbol{a}_3\,).$$

Since in the projective case the scene structure and the camera motion can only be recovered up to an arbitrary transformation, we can postmultiply the three matrices by the matrix $\mathcal{H}$ defined in the previous section. We obtain

$$\mathcal{M}_1 = (\,\text{Id} \quad 0\,), \quad \mathcal{M}_2 = (\,\mathcal{B}_2 \quad \boldsymbol{b}_2\,) \quad \text{and} \quad \mathcal{M}_3 = (\,\mathcal{B}_3 \quad \boldsymbol{b}_3\,),$$

where, similar to the previous section, we have $\boldsymbol{b}_i = \beta \boldsymbol{a}_i$ and $\mathcal{B}_i = \mathcal{A}_i + \frac{1}{\beta}\boldsymbol{b}_i \boldsymbol{\alpha}^T$ for $i = 2, 3$.

Under this transformation, $\boldsymbol{b}_2$ and $\boldsymbol{b}_3$ can still be interpreted as the homogeneous image coordinates of the epipoles e_{12} and e_{13}, and the trilinear constraints of Chapter 11 still hold, with the trifocal tensor defined this time by the three matrices

$$\mathcal{G}_1^i = \boldsymbol{b}_2 \boldsymbol{B}_3^{iT} - \boldsymbol{B}_2^i \boldsymbol{b}_3^T, \tag{14.3.2}$$

and $\boldsymbol{B}_2^i$ and $\boldsymbol{B}_3^i$ $(i = 1, 2, 3)$ denote the columns of $\mathcal{B}_2$ and $\mathcal{B}_3$.

Assuming that the trifocal tensor has been estimated from point or line correspondences as described in Chapter 11, our goal in this section is to recover the projection matrices $\mathcal{M}_2$ and $\mathcal{M}_3$. Let us first observe that

$$(\boldsymbol{b}_2 \times \boldsymbol{B}_2^i)^T \mathcal{G}_1^i = [(\boldsymbol{b}_2 \times \boldsymbol{B}_2^i)^T \boldsymbol{b}_2]\boldsymbol{B}_3^{iT} - [(\boldsymbol{b}_2 \times \boldsymbol{B}_2^i)^T \boldsymbol{B}_2^i]\boldsymbol{b}_3^T = \boldsymbol{0},$$

and, likewise,

$$\mathcal{G}_1^i(\boldsymbol{b}_3 \times \boldsymbol{B}_3^i) = [\boldsymbol{B}_3^{iT}(\boldsymbol{b}_3 \times \boldsymbol{B}_3^i)]\boldsymbol{b}_2 - [\boldsymbol{b}_3^T(\boldsymbol{b}_3 \times \boldsymbol{B}_3^i)]\boldsymbol{B}_2^i = \boldsymbol{0}.$$

It follows that the matrix $\mathcal{G}_1^i$ is singular (a fact already mentioned in Chapter 11) and that the vectors $\boldsymbol{b}_2 \times \boldsymbol{B}_2^i$ and $\boldsymbol{b}_3 \times \boldsymbol{B}_3^i$ lie respectively in its left and right nullspaces. In turn, this means that, once the trifocal tensor is known, we can compute the epipole $\boldsymbol{b}_2$ (resp. $\boldsymbol{b}_3$) as the common normal to the left (resp. right) nullspaces of the matrices $\mathcal{G}_1^i$ $(= i = 1, 2, 3)$ [**?**].

Once the epipoles are known, writing (14.3.2) for $i = 1, 2, 3$ provides 27 homogeneous linear equations in the 18 unknown entries of the matrices $\mathcal{B}_j$ $(j = 2, 3)$. These equations can be solved up to scale using linear least squares. Alternatively, it is possible to estimate the matrices $\mathcal{B}_j$ directly from the trilinear constraints associated with pairs of matching points or lines by writing the trifocal tensor coefficients as functions of these matrices, which leads once again to a linear estimation process.

Neither of these methods determines the matrices $\mathcal{M}_1$ and $\mathcal{M}_2$ uniquely of course. If desired, this ambiguity can be eliminated by imposing various constraints on the vectors $\boldsymbol{b}_2$ and $\boldsymbol{b}_3$. For example, Hartley [**?**] suggests imposing that $\boldsymbol{b}_2$ be a unit vector orthogonal to the columns of $\mathcal{B}_2$, which can be achieved as in the last section for an appropriate choice for $\boldsymbol{\alpha}$.

Once the projection matrices have been recovered, the projective structure of the scene can be recovered as well by using the perspective projection equations as linear constraints on the homogeneous coordinate vectors of the observed points and lines.

14.4 Motion Estimation from Multiple Views

Section 14.3 used the epipolar and trifocal constraints to reconstruct the camera motion and the corresponding scene structure from a pair or a triple of images.

Likewise, the quadrifocal tensor introduced in Chapter 11 can in principle be used to estimate the projection matrices associated with four cameras and the corresponding projective scene structure. However, multilinear constraints do not provide a direct method for handling $m > 4$ views in a uniform manner. Instead, the structure and motion parameters estimated from pairs, triples, or quadruples of successive views must be stitched together iteratively, as described for example in [**?**; **?**]. We now present an alternative where all images are taken into account at once in a non-linear optimization scheme.

14.4.1 A Factorization Approach to Projective Motion Analysis

In this section we present a factorization method for motion analysis due to Mahamud and Hebert [**?**] that generalizes the algorithm of Tomasi and Kanade presented in Chapter 13 to the projective case. Given m images of n points we can rewrite (14.0.1) as

$$\mathcal{D} = \mathcal{M}\mathcal{P}, \tag{14.4.1}$$

where

$$\mathcal{D} \stackrel{\text{def}}{=} \begin{pmatrix} z_{11}\boldsymbol{p}_{11} & z_{12}\boldsymbol{p}_{12} & \dots & z_{1n}\boldsymbol{p}_{1n} \\ z_{21}\boldsymbol{p}_{21} & z_{22}\boldsymbol{p}_{22} & \dots & z_{2n}\boldsymbol{p}_{2n} \\ \text{sec}\dots & \dots & \dots & \dots \\ z_{m1}\boldsymbol{p}_{m1} & z_{m2}\boldsymbol{p}_{m2} & \dots & z_{mn}\boldsymbol{p}_{mn} \end{pmatrix}, \quad \mathcal{M} \stackrel{\text{def}}{=} \begin{pmatrix} \mathcal{M}_1 \\ \mathcal{M}_2 \\ \dots \\ \mathcal{M}_m \end{pmatrix} \quad \text{and } \mathcal{P} \stackrel{\text{def}}{=} (\,\boldsymbol{P}_1\ \boldsymbol{P}_2\ \dots\ \boldsymbol{P}_n\,).$$

In particular, the $3m \times n$ matrix $\mathcal{D}$ has (at most) rank 4, thus if the projective depths z_{ij} were known, we could compute $\mathcal{M}$ and $\mathcal{P}$, just as in the affine case, by using singular value decomposition to factor $\mathcal{D}$. On the other hand, if $\mathcal{M}$ and $\mathcal{P}$ were known, we could read out the values of the projective depths z_{ij} from (14.4.1). This suggests an iterative scheme for estimating the unknowns z_{ij}, $\mathcal{M}$ and $\mathcal{P}$ by alternating steps where some of these unknowns are held constant while others are estimated. This section proposes such a scheme and shows that is guaranteed to converge to a local minimum of a physically-significant objective function.

Ideally, we would like to minimize the mean-squared distance between the observed image points and the point positions predicted from the parameters z_{ij}, $\mathcal{M}_i$ and $\mathcal{P}_j$, i.e.,

$$E = \frac{1}{mn}\sum_{i,j} |\boldsymbol{p}_{ij} - \frac{1}{z_{ij}}\mathcal{M}_i\boldsymbol{P}_j|^2.$$

Unfortunately the corresponding optimization problem is difficult since the error we are trying to minimize is highly non-linear in the unknowns z_{ij}, $\mathcal{M}_i$ and $\boldsymbol{P}_j$. Instead, let us define the vectors $\boldsymbol{d}_j = (z_{1j}\boldsymbol{p}_{1j}, \dots, z_{mj}\boldsymbol{p}_{mj})^T$ and $\boldsymbol{z}_j = (z_{1j}, \dots, z_{mj})^T$ $(j = 1, \dots, n)$, and minimize

$$E = \frac{1}{mn}E_j, \quad \text{where} \quad E_j \stackrel{\text{def}}{=} \frac{1}{|\boldsymbol{d}_j|^2}|\boldsymbol{d}_j - \mathcal{M}\boldsymbol{P}_j|^2$$

with respect to the unknowns $\mathcal{M}$, $\boldsymbol{z}_j$ and $\boldsymbol{P}_j$. Note that the vectors $\boldsymbol{z}_j$, $\boldsymbol{d}_j$ and $\boldsymbol{P}_j$ are defined up to a common scale factor. The normalizing factor $1/|\boldsymbol{d}_j|^2$ in E_j is used to avoid the trivial solution where all three are zero.

As proposed earlier, we will minimize E iteratively by alternating steps where motion paramaters are held constant while structure parameters are estimated and vice versa.

Let us assume that we are at some stage of this minimization process, fix the value of $\mathcal{M}$ to its current estimate and compute, for $j = 1, \ldots, n$, the values of $\boldsymbol{z}_j$ and $\boldsymbol{P}_j$ that minimize E_j. These values will of course minimize E as well.

Just as in the affine case discussed in Chapter 13, writing that the gradient of E_j with respect to the vector $\boldsymbol{P}_j$ is zero yields $\boldsymbol{P}_j = \mathcal{M}^\dagger \boldsymbol{d}_j$, where $\mathcal{M}^\dagger \stackrel{\text{def}}{=} (\mathcal{M}^T\mathcal{M})^{-1}\mathcal{M}^T$ is the pseudoinverse of $\mathcal{M}$. Substituting this value in the definition of E_j yields

$$E_j = \frac{1}{|\boldsymbol{d}_j|^2}|(\mathrm{Id} - \mathcal{M}\mathcal{M}^\dagger)\boldsymbol{d}_j|^2.$$

As noted in Chapter 13, the matrix $\mathcal{M}\mathcal{M}^\dagger$ associates with any vector in $\mathbb{R}^{3m}$ its orthogonal projection onto the subspace $V_{\mathcal{M}}$ spanned by the columns of $\mathcal{M}$. It follows immediately that minimizing E_j with respect to $\boldsymbol{z}_j$ and $\boldsymbol{P}_j$ is equivalent to minimizing the squared norm of the difference between $\boldsymbol{d}_j$ and its projection onto $V_{\mathcal{M}}$ under the constraint that $\boldsymbol{d}_j$ has unit length.

Now, $\mathcal{M}$ is a $3m \times 4$ matrix of rank 4, whose singular value decomposition $\mathcal{U}\mathcal{W}\mathcal{V}^T$ is formed by the product of a column-orthogonal $3m \times 4$ matrix $\mathcal{U}$, a 4×4 non-singular diagonal matrix $\mathcal{W}$ and a 4×4 orthogonal matrix $\mathcal{V}^T$. The pseudoinverse of $\mathcal{M}$ is $\mathcal{M}^\dagger = \mathcal{V}\mathcal{W}^{-1}\mathcal{U}^T$, and substituting this value in the expression of E_j immediately yields

$$E_j = \frac{1}{|\boldsymbol{d}_j|^2}|[\mathrm{Id} - \mathcal{U}\mathcal{U}^T]\boldsymbol{d}_j|^2 = \frac{1}{|\boldsymbol{d}_j|^2}[|\boldsymbol{d}_j|^2 - \boldsymbol{d}_j^T(\mathcal{U}\mathcal{U}^T)\boldsymbol{d}_j] = 1 - \frac{1}{|\boldsymbol{d}_j|^2}\boldsymbol{d}_j^T(\mathcal{U}\mathcal{U}^T)\boldsymbol{d}_j.$$

In turn, this means that minimizing E_j with respect to $\boldsymbol{z}_j$ and $\boldsymbol{P}_j$ is equivalent to maximizing

$$\frac{\boldsymbol{d}_j^T(\mathcal{U}\mathcal{U}^T)\boldsymbol{d}_j}{|\boldsymbol{d}_j|^2}$$

with respect to $\boldsymbol{z}_j$.

Observing that

$$\boldsymbol{d}_j^T = \boldsymbol{z}_j^T\mathcal{Q}_j, \quad \text{where} \quad \mathcal{Q}_j \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{p}_{1j}^T & 000 & \ldots & 000 \\ 000 & \boldsymbol{p}_{2j}^T & \ldots & 000 \\ \ldots & \ldots & \ldots & \ldots \\ 000 & 000 & \ldots & \boldsymbol{p}_{mj}^T \end{pmatrix},$$

finally shows that minimizing E_j is equivalent to maximizing

$$\frac{\boldsymbol{z}_j^T\mathcal{A}_j\boldsymbol{z}_j}{\boldsymbol{z}_j^T\mathcal{B}_j\boldsymbol{z}_j}$$

with respect to $\boldsymbol{z}_j$, with $\mathcal{A}_j \overset{\text{def}}{=} \mathcal{Q}_j \mathcal{U}\mathcal{U}^T \mathcal{Q}_j^T$ and $\mathcal{B}_j \overset{\text{def}}{=} \mathcal{Q}_j \mathcal{Q}_j^T$. This is a generalized eigenvalue problem, whose solution is the generalized eigenvector associated with the largest eigenvalue. If desired, the value of $\boldsymbol{P}_j$ can be computed at this stage as $\boldsymbol{P}_j = \mathcal{M}\mathcal{M}^\dagger \boldsymbol{d}_j = \mathcal{U}\mathcal{U}^T \boldsymbol{d}_j$.

Let us now fix $\boldsymbol{z}_j$ (hence $\boldsymbol{d}_j$), and minimize E with respect to $\mathcal{M}$ and the vectors $\boldsymbol{P}_j$.Writing that the gradient of E with respect to $\boldsymbol{P}_j$ is zero yields once again $\boldsymbol{P}_j = \mathcal{M}\mathcal{M}^\dagger \boldsymbol{d}_j$, thus

$$E = \sum_{j=1}^{n} \frac{1}{|\boldsymbol{d}_j|^2} |(\mathrm{Id} - \mathcal{M}\mathcal{M}^\dagger)\boldsymbol{d}_j|^2 = \sum_{j=1}^{n} |(\mathrm{Id} - \mathcal{M}\mathcal{M}^\dagger)\tilde{\boldsymbol{d}}_j|^2,$$

where $\tilde{\boldsymbol{d}}_j \overset{\text{def}}{=} \dfrac{1}{|\boldsymbol{d}_j|}\boldsymbol{d}_j$.

In other words, E measures the mean squared norm of the difference between the unit vectors $\tilde{\boldsymbol{d}}_j$ and their orthogonal projections onto the subspace $V_{\mathcal{M}}$ spanned by the columns of $\mathcal{M}$. As in the affine case, we now use Theorem 4, that states that E is minimum when $V_{\mathcal{M}}$ is the range of the matrix $\tilde{\mathcal{U}}_4$ formed by the four leftmost columns of $\tilde{\mathcal{U}}$, where $\tilde{\mathcal{U}}\tilde{\mathcal{W}}\tilde{\mathcal{V}}^T$ denotes the singular value decomposition of the matrix $\tilde{\mathcal{D}}$ whose columns are the unit vectors $\tilde{\boldsymbol{d}}_j$. In particular, the matrix $\mathcal{M}$ minimizing E verifies $V_{\mathcal{M}} = V_{\tilde{\mathcal{U}}_4}$ and we can take $\mathcal{M} = \tilde{\mathcal{U}}_4$.

This yields the iterative procedure sketched below. Note that this procedure does not explicitly maintain a separate copy of $\tilde{\mathcal{D}}$. Instead, the columns of the matrix $\mathcal{D}$ are normalized at each iteration.

```
Repeat:
    1. normalize each column of the data matrix $\mathcal{D}$;
    2. compute the singular value decomposition $\mathcal{U}\mathcal{W}\mathcal{V}^T$ of the matrix $\mathcal{D}$, and set
       $\mathcal{U}_4$ to be the $3m \times 4$ matrix formed by the four leftmost columns of $\mathcal{U}$;
    3. for $j = 1, \ldots, n$ do:
        (a) compute $\mathcal{A}_j = \mathcal{Q}_j \mathcal{U}_4 \mathcal{U}_4^T \mathcal{Q}_j$ and $\mathcal{B}_j = \mathcal{Q}_j \mathcal{Q}_j^T$;
        (b) solve the generalized eigenvalue problem $\mathcal{A}_j \boldsymbol{z} = \lambda \mathcal{B}_j \boldsymbol{z}$ and set $\boldsymbol{z}_j$ to be
            the generalized eigenvector associated with the largest eigenvalue;
        (c) update the corresponding column of $\mathcal{D}$;
until convergence.
```

Algorithm 14.1: *A factorization algorithm for projective shape from motion.*

It should be noted that this algorithm is guaranteed to converge to some local minimum of the error function E. Indeed, let E_0 be the current error value at the beginning of each iteration; the first two steps of the algorithm do not change the vectors $\boldsymbol{z}_j$ but minimizes E with respect to the unknowns $\mathcal{M}$ and $\boldsymbol{P}_j$. If E_2 is the value of the error at the end step 2, we have therefore $E_2 \leq E_0$. Now step 3 does not change the matrix $\mathcal{M}$ but minimizes each error term E_j with respect to both the vectors $\boldsymbol{z}_j$ and $\boldsymbol{P}_j$. Therefore the error E_3 at the end of this step is smaller than or equal to E_2. This shows that the error decreases in a monotone manner at each iteration, and since it is bounded below by zero, we conclude that the process converges to a local minimum of E.

Whether this local minimum will turn out to be the global one depends, of course, on the choice of initial values chosen for the various unknown parameters. A possible choice, used in the experiments presented in [**?**], is to initialize the projective depths z_{ij} to 1, which effectively amounts to starting with a weak-perspective projection model. The authors also report that the data preprocessing suggested by Hartley [**?**] and already used in the normalized eight-point algorithm for weak calibration described in Chapter 11 improves the robustness of the algorithm. Figure 14.7(a) shows two images in a sequence of 20 pictures of an outdoor scene. A total of 30 points were tracked manually across the sequence, with a localization error of ∓1 pixel. Figure 14.7(b) plots the evolution of the average and maximum error between the observed and predicted image point positions when various subsets of the image sequence are used for training and testing.

14.4.2 Bundle Adjustment

Given initial estimates for the matrices $\mathcal{M}_i$ $(i = 1, \ldots, m)$ and vectors $\boldsymbol{P}_j$ $(j = 1, \ldots, n)$, we can refine these estimates by using non-linear least squares to minimize the global error measure

$$E = \frac{1}{mn} \sum_{i,j} [(u_{ij} - \frac{\boldsymbol{m}_{i1} \cdot \boldsymbol{P}_j}{\boldsymbol{m}_{i3} \cdot \boldsymbol{P}_j})^2 + (v_{ij} - \frac{\boldsymbol{m}_{i2} \cdot \boldsymbol{P}_j}{\boldsymbol{m}_{i3} \cdot \boldsymbol{P}_j})^2].$$

This is the method of *bundle adjustment*, whose name originates from the field of photogrammetry. Although it may be expensive, it offers the advantage of combining all measurements to minimize a physically-significant error measure, namely the mean-squared error between the actual image point positions and those predicted using the estimated scene structure and camera motion.

14.5 From Projective to Euclidean Structure and Motion

Although projective structure is useful by itself, in most cases it is the Euclidean, or metric structure of the scene which is the true object of interest. Let us assume that one of the techniques presented in Section 14.4 has been used to estimate the projection matrices $\mathcal{M}_i$ $(i = 1, \ldots, m)$ and the point positions $\boldsymbol{P}_j$ $(j = 1, \ldots, n)$ from m images of these points. We know that any other reconstruction *and in particular a metric one* will be separated from this one by a projective transformation. In other words, if $\hat{\mathcal{M}}_i$ and $\hat{\boldsymbol{P}}_j$ denote the metric shape and motion parameters,

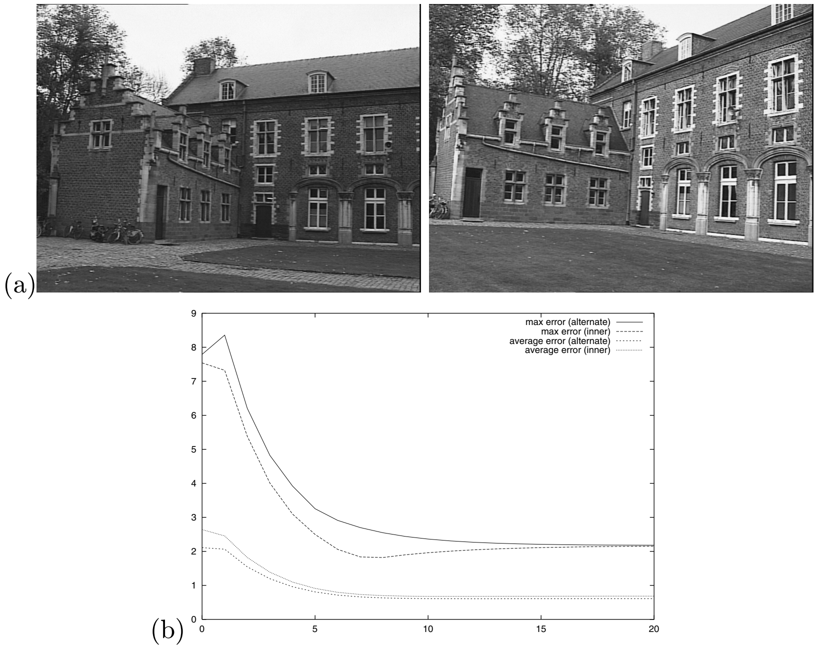


Figure 14.7. Iterative projective estimation of camera motion and scene structure: (a) the first and last images in the sequence; (b) plot of the average and maximum reprojection error as a function of iteration number. Two experiments were conducted: in the first one (alternate) alternate images in the sequence are used as training and testing datasets; in the second experiment (inner), the first five and last five pictures were used as training set, and the remaining images were used for testing. In both cases, the average error falls below 1 pixel after 15 iterations. Reprinted from [**?**], Figure 4.

there exists a 4×4 matrix $\mathcal{Q}$ such that $\hat{\mathcal{M}}_i = \mathcal{M}_i \mathcal{Q}$ and $\hat{\boldsymbol{P}}_j = \mathcal{Q}^{-1} \boldsymbol{P}_j$ (the matrix $\mathcal{Q}$ is of course only defined up to an arbitrary similarity, i.e., rigid transformation plus scaling). This section presents a number of methods for computing the *metric upgrade* matrix $\mathcal{Q}$ and thus recovering the metric shape and motion from the projective ones.

14.5.1 Metric Upgrades from (Partial) Camera Calibration

It is a simple matter to adapt the affine method introduced in Chapter 13 to the projective setting when the intrinsic parameters of all cameras are known: indeed, the 3×3 matrix formed by the three leftmost columns of each matrix $\hat{\mathcal{M}}_i$ is in this case a rotation matrix scaled by an unknown factor. Writing that its rows are perpendicular to each other and have the same length yields

$$\begin{cases} \hat{\boldsymbol{m}}_{i1}^T\hat{\boldsymbol{m}}_{i2} = \boldsymbol{m}_{i1}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i2} = 0, \\ \hat{\boldsymbol{m}}_{i2}^T\hat{\boldsymbol{m}}_{i3} = \boldsymbol{m}_{i2}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i3} = 0, \\ \hat{\boldsymbol{m}}_{i3}^T\hat{\boldsymbol{m}}_{i1} = \boldsymbol{m}_{i3}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i1} = 0, \\ \hat{\boldsymbol{m}}_{i1}^T\hat{\boldsymbol{m}}_{i1} - \hat{\boldsymbol{m}}_{i2}^T\hat{\boldsymbol{m}}_{i2} = \boldsymbol{m}_{i1}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i1} - \boldsymbol{m}_{i2}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i2} = 0, \\ \hat{\boldsymbol{m}}_{i2}^T\hat{\boldsymbol{m}}_{i2} - \hat{\boldsymbol{m}}_{i3}^T\hat{\boldsymbol{m}}_{i3} = \boldsymbol{m}_{i2}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i2} - \boldsymbol{m}_{i3}^T\mathcal{Q}_3\mathcal{Q}_3^T\boldsymbol{m}_{i3} = 0, \end{cases} \tag{14.5.1}$$

where $\mathcal{Q}_3$ is the 4×3 matrix formed by the three leftmost columns of $\mathcal{Q}$. To determine $\mathcal{Q}$ uniquely, we can for example assume that the world coordinate system and the first camera's frame coincide. Given m images, we obtain 12 linear equations and $5(m-1)$ quadratic ones in the coefficients of $\mathcal{Q}$. These equations can be solved using non-linear least squares.

Alternatively, the constraints (14.5.1) are linear in the coefficients of the symmetric matrix $\mathcal{A} \stackrel{\text{def}}{=} \mathcal{Q}_3\mathcal{Q}_3^T$, allowing its estimation from at least two images via linear least squares. Note that $\mathcal{A}$ has rank 3, a constraint that is not enforced by our construction. To recover $\mathcal{Q}_3$, let us also note that since $\mathcal{A}$ is symmetric, it can be diagonalized in an orthonormal basis as $\mathcal{A} = \mathcal{U}\mathcal{D}\mathcal{U}^T$, where $\mathcal{D}$ is the diagonal matrix formed by the eigenvalues of $\mathcal{A}$ and $\mathcal{U}$ is the orthogonal matrix formed by its eigenvectors. In the absence of noise, $\mathcal{A}$ is positive semidefinite with three positive and one zero eigenvalues, and $\mathcal{Q}_3$ can be computed as $\mathcal{U}_3\sqrt{\mathcal{D}_3}$, where $\mathcal{U}_3$ is the matrix formed by the columns of $\mathcal{U}$ inassociated with the positive eigenvalues of $\mathcal{A}$, and $\mathcal{D}_3$ is the corresponding sub-matrix of $\mathcal{D}$. Because of noise, however, $\mathcal{A}$ will usually have maximal rank, and its smallest eigenvalue may even be negative. As shown in the exercises, if we take this time $\mathcal{U}_3$ and $\mathcal{D}_3$ to be the sub-matrices of $\mathcal{U}$ and $\mathcal{D}$ associated with the three largest (positive) eigenvalues of $\mathcal{A}$, then $\mathcal{U}_3\mathcal{D}_3\mathcal{U}_3^T$ provides the best positive semidefinite rank-3 approximation of $\mathcal{A}$ in the sense of the Frobenius norm,[3] and we can take as before $\mathcal{Q}_3 = \mathcal{U}_3\sqrt{\mathcal{D}_3}$.

This method can easily be adapted to the case where only some of the intrinsic camera parameters are known: let us write the metric upgrade matrix as $\mathcal{Q} = (\mathcal{Q}_3, \boldsymbol{q}_4)$, where $\mathcal{Q}_3$ is as before a 4×3 matrix and $\boldsymbol{q}_4$ is a vector in $\mathbb{R}^4$. We can rewrite the equation $\hat{\mathcal{M}}_i = \mathcal{M}_i\mathcal{Q}$ as

$$\mathcal{M}_i(\mathcal{Q}_3, \boldsymbol{q}_4) = \mathcal{K}_i(\mathcal{R}_i, \boldsymbol{t}_i) \Longrightarrow \mathcal{M}_i\mathcal{Q}_3 = \mathcal{K}_i\mathcal{R}_i,$$

where $\mathcal{K}_i$, $\mathcal{R}_i$ and $\boldsymbol{t}_i$ denote respectively the matrix of intrinsic parameters, the rotation matrix and the translation vector associated with $\hat{\mathcal{M}}_i$.

Using the fact that $\mathcal{R}_i$ is an orthogonal matrix allows us to write

$$\mathcal{M}_i\mathcal{A}\mathcal{M}_i^T = \mathcal{K}_i\mathcal{K}_i^T. \tag{14.5.2}$$

Thus every image provides a set of constraints between the entries of $\mathcal{K}_i$ and $\mathcal{A}$. Assuming for example that the center of the image is known for each camera, we can write the square of the matrix $\mathcal{K}_i$ as

---

[3] Note the obvious similarity between this result and Theorem 3.

$$\mathcal{K}_i\mathcal{K}_i^T = \begin{pmatrix} \alpha_i^2 \dfrac{1}{\sin^2\theta_i} & -\alpha_i\beta_i \dfrac{\cos\theta_i}{\sin^2\theta_i} & 0 \\ -\alpha_i\beta_i \dfrac{\cos\theta_i}{\sin^2\theta_i} & \beta_i^2 \dfrac{1}{\sin^2\theta_i} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

In particular, the part of Equation (14.5.2) corresponding to the zero entries of $\mathcal{K}_i\mathcal{K}_i^T$ provides two independent linear equations in the ten coefficients of the 4×4 symmetrix matrix $\mathcal{A}$. With $m \geq 5$ images, these parameters can be estimated via linear least squares. Once $\mathcal{A}$ is known, $\mathcal{Q}$ can be estimated as before. Figure 14.8 shows a texture-mapped picture of the 3D model of a castle obtained by a variant of this method [**?**].

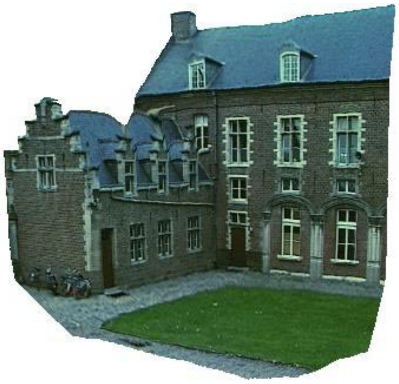


Figure 14.8. A synthetic texture-mapped image of a castle constructed via projective motion analysis followed by metric upgrade. The principal point is assumed to be known. Reprinted from [**?**], Figure 6.13.

14.5.2 Metric Upgrades from Minimal Assumptions

We now consider the case where the only constraint on the intrinsic parameters is that the pixels be rectangular, a condition satisfied (to a very good approximation) by all digital cameras. Theorem 2 in Chapter 5 shows that arbitrary 3×4 matrices are *not* zero-skew perspective projection matrices. It can therefore be hoped that better-than-projective reconstructions of the world can be achieved for zero-skew cameras. We will say that a projective transformation $\mathcal{Q}$ preserves zero skew when, for *any* zero-skew perspective projection matrix $\mathcal{M}$, the matrix $\mathcal{MQ}$ is also a zero-skew perspective projection matrix. Heyden and Åström [**?**] and Pollefeys *et al.* [**?**] have independently shown the following important result.

Theorem 5: *The class of transformations that preserve zero skew is the group of similarity transformations.*

The proof of this theorem is constructive: for example, Pollefeys *et al.* [?] exhibit a set of eight camera positions and orientations that constrain the transformation to be a similarity. Unfortunately, it does not provide a method for estimating the camera calibration parameters. We use in this section the elements of line geometry presented in Chapter 5[4] to derive a linear (or rather linearized) technique that exploits the zero-skew constraint to compute a metric upgrade of a projective reconstruction.

This section provides an algebraic and geometric characterization of the 4×4 matrices $\mathcal{Q}$ such that, if $\hat{\mathcal{M}} = \mathcal{M}\mathcal{Q}$, the rows of $\hat{\mathcal{M}}$ satisfy the condition of Theorem 2. We write the matrices $\hat{\mathcal{M}}$, $\mathcal{M}$ and $\mathcal{Q}$ as

$$\hat{\mathcal{M}} = \begin{pmatrix} \hat{\boldsymbol{m}}_1^T & \hat{m}_{14} \\ \hat{\boldsymbol{m}}_2^T & \hat{m}_{24} \\ \hat{\boldsymbol{m}}_3^T & \hat{m}_{34} \end{pmatrix}, \quad \mathcal{M} = \begin{pmatrix} \boldsymbol{m}_1^T \\ \boldsymbol{m}_2^T \\ \boldsymbol{m}_3^T \end{pmatrix} \quad \text{and} \quad \mathcal{Q} = (\, \boldsymbol{q}_1 \quad \boldsymbol{q}_2 \quad \boldsymbol{q}_3 \quad \boldsymbol{q}_4 \,).$$

Note that the vectors $\boldsymbol{m}_i$ and $\boldsymbol{q}_i$ are elements of $\mathbb{R}^4$ but the vectors $\hat{\boldsymbol{m}}_i$ are elements of $\mathbb{R}^3$. With this notation, we have the following result [?].

Theorem 6: *Given a projection matrix $\mathcal{M}$ and a projective transformation $\mathcal{Q}$, a necessary and sufficient condition for the matrix $\hat{\mathcal{M}} = \mathcal{M}\mathcal{Q}$ to satisfy the zero-skew constraint*

$$(\hat{\boldsymbol{m}}_1 \times \hat{\boldsymbol{m}}_3) \cdot (\hat{\boldsymbol{m}}_2 \times \hat{\boldsymbol{m}}_3) = 0$$

is that

$$\boldsymbol{\lambda}^T \mathcal{R}^T \mathcal{R} \boldsymbol{\mu} = 0, \tag{14.5.3}$$

where

$$\mathcal{R} \stackrel{\text{def}}{=} \begin{pmatrix} (\boldsymbol{q}_2 \wedge \boldsymbol{q}_3)^T \\ (\boldsymbol{q}_3 \wedge \boldsymbol{q}_1)^T \\ (\boldsymbol{q}_1 \wedge \boldsymbol{q}_2)^T \end{pmatrix}, \quad \boldsymbol{\lambda} \stackrel{\text{def}}{=} \boldsymbol{m}_1 \wedge \boldsymbol{m}_3 \quad \textit{and} \quad \boldsymbol{\mu} \stackrel{\text{def}}{=} \boldsymbol{m}_2 \wedge \boldsymbol{m}_3.$$

The proof of this theorem relies on elementary properties of the exterior product to show that $\hat{\boldsymbol{m}}_1 \times \hat{\boldsymbol{m}}_3 = \mathcal{R}\boldsymbol{\lambda}$ and $\hat{\boldsymbol{m}}_2 \times \hat{\boldsymbol{m}}_3 = \mathcal{R}\boldsymbol{\mu}$, from which the theorem immediately follows (see exercises).

A matrix $\mathcal{Q}$ satisfying (14.5.3) can be estimated from m images using linear methods: we first use linear least squares to estimate the matrix $\mathcal{S} \stackrel{\text{def}}{=} \mathcal{R}^T\mathcal{R}$, then take advantage of elementary properties of symmetric (but possibly indefinite) matrices to factor $\mathcal{S}$ and compute $\mathcal{R}$. Once $\mathcal{R}$ is known, it is a simple matter to determine the matrix $\mathcal{Q}$ using once again linear least squares. This approach linearizes the estimation process since (14.5.3) is an equation of degree 4 in the coefficients of $\mathcal{Q}$. It is easy to show (see exercises) that the columns of the matrices $\mathcal{R}$ and $\mathcal{S}$ are the Plücker coordinates of a family of lines and it follows that their entries satisfy a number of quadratic constraints as well as the linear constraint:

$$S_{16} - S_{25} + S_{34} = 0.$$

---

[4] This may be a good time to look back at that chapter to brush up on your line geometry.

We can now estimate the matrix $\mathcal{S}$. Let us first note that (14.5.3) is a linear constraint on the coefficients of $\mathcal{S}$, that can be rewritten as

$$\sum_{i=1}^{6} \lambda_i \mu_i S_{ii} + \sum_{1 \le i < j \le 6} (\lambda_i \mu_j + \lambda_j \mu_i) S_{ij} = 0 \tag{14.5.4}$$

where the coefficients λ_i and μ_i denote the coordinates of the vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ and the 20 coefficients S_{ij} denote the entries of $\mathcal{S}$. The unknown S_{16} can be eliminated by using the constraint $S_{16} - S_{25} + S_{34} = 0$ and the fact that, since the lines associated with the vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ both lie in the focal plane, we have $(\boldsymbol{\lambda}|\boldsymbol{\mu}) = 0$. This allows us to and rewrite (14.5.4) as

$$\sum_{i=1}^{6} \lambda_i \mu_i S_{ii} + \sum_{\substack{1 \le i < j \le 6 \\ i+j \ne 7}} (\lambda_i \mu_j + \lambda_j \mu_i) S_{ij} + a_{25} S_{25} + a_{34} S_{34} = 0, \tag{14.5.5}$$

where

$$\begin{cases} a_{25} = 2(\lambda_2\mu_5 + \lambda_5\mu_2) - (\lambda_3\mu_4 + \lambda_4\mu_3), \\ a_{34} = 2(\lambda_3\mu_4 + \lambda_4\mu_3) - (\lambda_2\mu_5 + \lambda_5\mu_2), \end{cases}$$

and the missing elements in the second sum in (14.5.5) correspond to the terms S_{16}, S_{25} and S_{34}.

With only 20 out of the 21 original unknown coefficients left, writing (25.5.1) for $m \ge 19$ images yields an overdetermined homogeneous system of linear equations of the form $\mathcal{A}\boldsymbol{s} = 0$, where $\mathcal{A}$ is an $m \times 20$ data matrix and $\boldsymbol{s}$ is the vector formed by the 20 independent coefficients of $\mathcal{S}$. The least-squares solution of this system is computed (up to an irrelevant scale factor) using the techniques described in Chapter 5. The S_{16} entry is then computed as $S_{25} - S_{34}$. Note that this linear process ignores the quadratic equations satisfied by the entries of the matrix $\mathcal{S}$. Once the symmetric matrix $\mathcal{S} = \mathcal{R}^T\mathcal{R}$ is known, it can be used to estimate the rank-3 matrix $\mathcal{R}^T$ via eigenvalue decomposition, just as in the previous section.

Once the matrix $\mathcal{R}$ is known, we can recover the vectors $\boldsymbol{q}_i$ $(i = 1, 2, 3)$ from $\mathcal{R}$ using linear least squares, exactly as we recovered the vectors $\boldsymbol{m}_i$ from the matrix $\tilde{\mathcal{M}}$ in Chapter 5. Once the vectors $\boldsymbol{q}_i$ are known, we can complete the construction of $\mathcal{Q}$ by imposing, for example, that the optical center of the first camera be used as origin of the world coordinate system. This translates into the fourth column of $\hat{\mathcal{M}}_1$ being zero, and allows us to compute $\boldsymbol{q}_4$ (up to scale) as the solution of $\mathcal{M}_1\boldsymbol{q}_4 = 0$. This unknown scale factor reflects the fact that absolute scale cannot be recovered from images.

Let us conclude by noting that, given m projection matrices $\mathcal{M}_i$, the estimates of the vectors $\boldsymbol{q}_i$ $(i = 1, 2, 3)$ obtained from the linear least-squares process can be refined using non-linear least-squares to minimize the average squared skew of the projection matrices, i.e.,

$$\frac{1}{m} \sum_{i=1}^{m} \left[\arcsin \frac{(\mathcal{R}\boldsymbol{\lambda}_i) \cdot (\mathcal{R}\boldsymbol{\mu}_i)}{|\mathcal{R}\boldsymbol{\lambda}_i|\,|\mathcal{R}\boldsymbol{\mu}_i|} \right]^2,$$

with respect to the vectors $\boldsymbol{q}_i$ $(i = 1, 2, 3)$. The vector $\boldsymbol{q}_4$ can then be computed as before.

Figure 14.9(a) shows two views of the projective reconstruction of a desk scene featuring a volleyball and a cylindrical box [?]. The data consists of 182 projection matrices and 3506 points. Applying the method described in this section to this reconstruction yields the results shown in Figure 14.9(b)-(c). The skew averaged over the 182 input matrices is 5.68° after the linear stage of the algorithm, and 0.46° after its non-linear stage.

14.6 Notes

The short introduction to projective geometry given at the beginning of this chapter focuses on the analytical side of things. See for example [?; ?; ?] for thorough introductions to analytical projective geometry and [?] for an axiomatic presentation. Projective structure from motion is covered in detail in the excellent book by Hartley and Zisserman [?].

As mentioned by Faugeras [?], the problem of calculating the epipoles and the epipolar transformations compatible with seven point correspondences was first posed by Chasles [?] and solved by Hesse [?]. The problem of estimating the epipolar geometry from five point correspondences for internally calibrated cameras was solved by Kruppa [?]. An excellent modern account of Hesse's and Kruppa's techniques can be found in [?], where the *absolute conic*, an imaginary conic section invariant through similarities, is used to derive two tangency constraints that make up for the missing point correspondences. These methods are of course mostly of theoretical interest since their reliance on a minimal number of correspondences limits their ability to deal with noise. The weak-calibration methods of Luong *et al.* [?; ?] and Hartley [?] described in Chapter 11 provide reliable and accurate alternatives.

Faugeras [?] and Hartley *et al.* [?] introduced independently the idea of using a pair of uncalibrated cameras to recover the projective structure of a scene. Other notable work in this area includes, for example, [?; ?]. Section 14.2.1 presents Faugeras' original method, and its geometric variant presented in Section 14.2.2 is taken from [?]. Hartley [?; ?; ?] developed the two- and three-view motion analysis techniques also presented in this chapter.

An iterative algorithm for perspective motion and structure recovery using calibrated cameras is given in [?]. The extension of factorization approaches to structure and motion recovery was first proposed by Sturm and Triggs [?]. Its variant due to Mahamud and Hebert [?] presented in this chapter has the advantage of having provable convergence.

The problem of computing metric upgrades of projective reconstructions when some of the intrinsic parameters are known has been addressed by a number of authors (e.g., [?; ?; ?]). The matrix $\mathcal{A} = \mathcal{Q}_3 \mathcal{Q}_3^T$ introduced in Section 14.5 can be interpreted geometrically as the projective representation of the dual of the absolute conic, the *absolute dual quadric* [?]. Like the absolute conic, this quadric surface

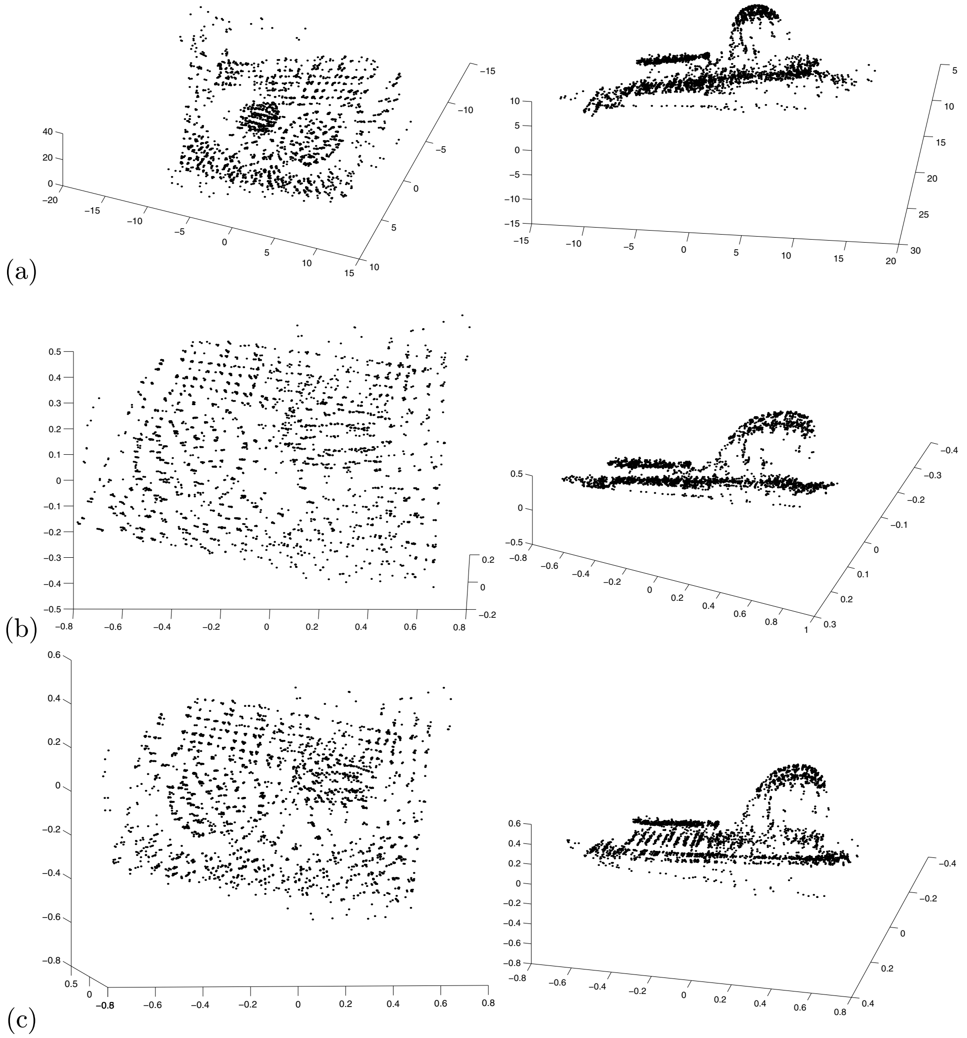


Figure 14.9. Computing metric upgrades of projective reconstructions: (a) two views of the projective reconstruction of a simple scene, and its metric upgrades after (b) the linear stage of the method presented in this section, and (c) its non-linear refinement stage. Reprinted from [**?**], Figure 1.

is invariant through similarities, and the (dual) conic section associated with $\mathcal{K}_i\mathcal{K}_i^T$ is simply the projection of this quadric surface into the corresponding image. Self-calibration is the process of computing the intrinsic parameters of a camera from point correspondences with unknown Euclidean positions. Work in this area was pioneered by Faugeras and Maybank [**?**] for cameras with fixed intrinsic parameters. A number of reliable self-calibration methods are now available [**?**; **?**; **?**], and they can of course also be used to upgrade projective reconstructions to metric ones. The problem of computing metric upgrades of projective reconstructions under minimal camera constraints such as a zero skew was first addressed by Heyden and Åström [**?**] and Pollefeys *et al.* [**?**]. The method discussed in Section 14.5 was proposed in [**?**]. See also [**?**] for a related approach.

14.7 Assignments

Exercises

1. Show that the perspective projection mapping between two planes of $\mathbb{P}^3$ is a projective transformation (Example 14.4).

   Hint: If Π and Π' denote the two planes of $\mathbb{P}^3$ under consideration, and V and V' denote the two corresponding planes of $\mathbb{R}^3$, i.e., $\Pi = P(V)$ and $\Pi' = P(V')$, construct the mapping f associated with the restriction to Π of the linear projection onto V' in the direction $p^{-1}(O)$.

2. In this exercise you will show that the cross-ratio of four collinear points A, B, C and D is equal to

   $$[A, B, C, D] = \frac{\sin(\beta+\gamma)\sin\gamma}{\sin(\alpha+\beta)\sin\alpha},$$

   where the angles α, β and γ are defined as in Figure 14.2.

   (a) Show that the area of a triangle PQR is

   $$A(P, Q, R) = PQ \times RH = PQ \times PR \sin\theta,$$

   where PQ denotes the distance between the two points P and Q, H is the projection of R onto the line passing through P and Q, and θ is the angle between the lines joining the point P to the points Q and R.

   (b) Define the ratio of three collinear points A, B, C as

   $$R(A, B, C) = \frac{\overline{AB}}{\overline{BC}}$$

   for some orientation of the line supporting the three points. Show that $(A, B, C) = A(A, B, O)/A(B, C, O)$ where O is some point not lying on this line.

(c) Conclude that the cross-ratio $[A, B, C, D]$ is indeed given by the formula above.

3. Show that a collineation between two pencils of lines can be written as

$$\tau \to \tau' = \frac{a\tau + b}{c\tau + d},$$

where τ and τ' are the slopes of the lines.

Hint: Parameterize each pencil of lines by the vertical and horizontal lines in the pencil.

Solution: Consider the first pencil and let a denote the common point of its lines, with (non-homogeneous) coordinates (α, β) in some fixed coordinate system. The vertical and horizontal lines in the pencil have coordinates $(1, 0, -\alpha)$ and $(0, 1, -\beta)$. In particular, any line in the pencil can be written as $(x, y, -\alpha x - \beta y)$, where x and y are homogeneous coordinates defined up to scale. If the line passes through the point (u, v) we have $(u-\alpha)x + (v-\beta)y = 0$ or $x = \beta - v$ and $y = u - \alpha$. Using the same construction for the second line and writing the collineation in matrix form yields

$$\begin{pmatrix} \beta' - v' \\ u' - \alpha' \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \beta - v \\ u - \alpha \end{pmatrix},$$

or

$$\frac{v' - \beta'}{u' - \alpha'} = -\frac{A(v-\beta) - B(u-\alpha)}{C(v-\beta) - D(u-\alpha)},$$

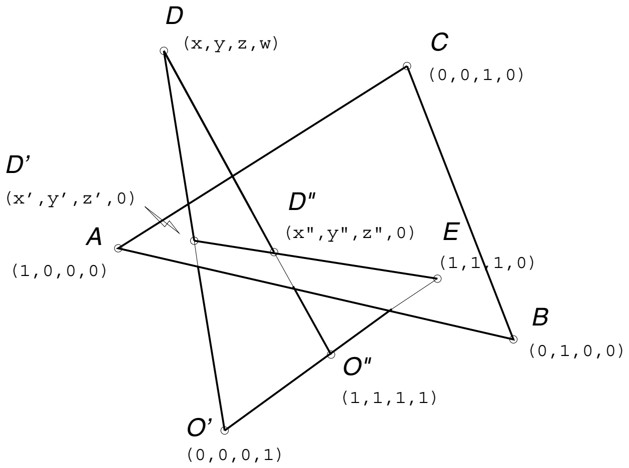
and the result follows immediately by taking $a = -A$, $b = B$, $c = C$ and $d = -D$.

4. Show that the fundamental matrix $\mathcal{F}$ can be expressed as

$$\mathcal{F} = \begin{pmatrix} b & a & -a\beta - b\alpha \\ -d & -c & c\beta + d\alpha \\ d\beta' - b\alpha' & c\beta' - a\alpha' & -c\beta\beta' - d\beta'\alpha + a\beta\alpha' + b\alpha\alpha' \end{pmatrix},$$

where (α, β) and (α', β') denote the coordinates of the epipoles, and a, b, c and d denote the coefficients of the epipolar transformation.

5. **Rewrite this as a true exercise.** Here we revisit the three-point reconstruction problem in the context of the *homogeneous* coordinates of the point D in the projective basis formed by the tetrahedron (A, B, C, O') and the unit point O''. Note that the ordering of the reference points, and thus the ordering of the coordinates, is different from the one used earlier: this new choice is, like the previous one, made to facilitate the reconstruction.



With this choice of coordinates, the point E where the baseline intersects the plane ABC has coordinates $(1,1,1,0)$. We denote the (unknown) coordinates of the point D by (x,y,z,w), and equip the first (resp. second) image plane with the triangle of reference a',b',c' (resp. a'',b'',c'') and the unit point e' (resp. e''), and denote by (x',y',z') (resp. (x'',y'',z'')) the coordinates of the point d' (resp. d'').

Obviously, the coordinates of the points D' and D'' are simply $(x',y',z',0)$ and $(x'',y'',z'',0)$. It remains to compute the coordinates of D as the intersection of the two rays $O'D'$ and $O''D''$.

We write $D=\lambda'O'+\mu'D'=\lambda''O''+\mu''D''$, which yields:

$$\begin{cases} x=\mu'x'=\lambda''+\mu''x'', \\ y=\mu'y'=\lambda''+\mu''y'', \\ z=\mu'z'=\lambda''+\mu''z'', \\ w=\lambda'=\lambda''. \end{cases} \tag{14.7.1}$$

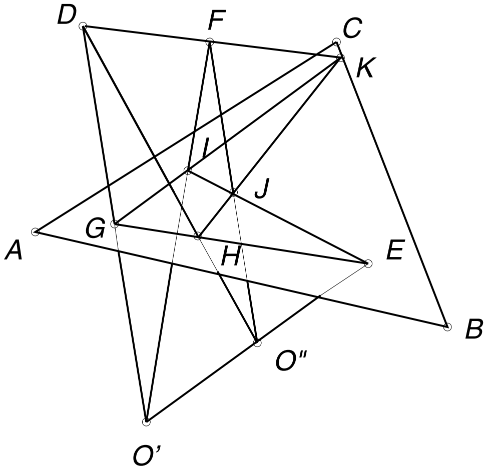
The values of μ',μ'',λ'' are found (up to some scale factor) by solving the following homogeneous system:

$$\begin{pmatrix} -x' & x'' & 1 \\ -y' & y'' & 1 \\ -z' & z'' & 1 \end{pmatrix} \begin{pmatrix} \mu' \\ \mu'' \\ \lambda'' \end{pmatrix} = 0. \tag{14.7.2}$$

Note that the determinant of this equation must be zero, which corresponds to D',D'', and E being collinear. In practice, (14.7.2) is solved through linear least-squares, and the values of x,y,z,w are then computed using (14.7.1).

6. **Rewrite this as a true exercise.** This exercise gives a geometric projective scene reconstruction method when the projective frame is formed by five arbitrary points A,B,C,D,F. Here, we give a geometric construction of the

points O', O''. Scene points can then be reconstructed using the three-point approach. We change the notation a bit to avoid confusion. While E is still the intersection of the baseline with the ABC plane, the rays $O'D$, $O''D$, $O'F$, and $O''F$ now respectively intersect this plane in G, H, I, and J. In addition, the line DF intersects the plane in K, so this point is known as well.



The position of the projection of a point P in the first (resp. second) image is as before denoted by p' (resp. p''). (Clearly, $g' = d'$, $i' = f'$, $h'' = d''$, and $j'' = f''$.)

We use the obvious fact that the points a', b',c', e', d', h', f', j', k' and a'', b'', c'', e'', g'', d'', i'', f'', k'' are in projective correspondence. Note that h', j', g'', i'' are not directly observable in the images. However, we can measure the projective coordinates of $d' = g'$ and $i' = f'$ in the a', b', c', e' basis, and thus reconstruct g'', i'' in the a'', b'', c'', e'' basis. This yields the point k'' as the intersection of the lines $g''i''$ and $d''f'' = h''j''$.

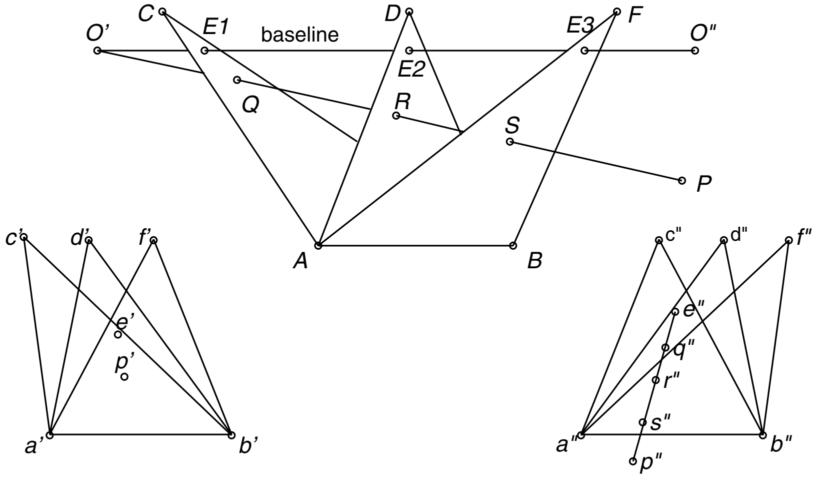
We now use the obvious projective correspondence between the points a'', b'', c'', g'', i'', $d'' = h''$, $f'' = j''$, k'' and A, B, C, G, I, H, J, K. From the projective coordinates of g'', h'', i'', j'' in the a'', b'', c'', k'' basis, we reconstruct the points G, H, I, J in the A, B, C, K basis. The final step of the reconstruction yields the point O' as the intersection of the lines DG and FI, and the point O'' as the intersection of the lines DH and FJ.

The point P can also be reconstructed directly: we use the C, A, B, D, F basis and the following cross-ratios

$$\begin{cases} k_0 = [ABC, ABD, ABF, ABP], \\ k_1 = [BCA, BCD, BCF, BCP], \\ k_2 = [CAB, CAD, CAF, CAP]. \end{cases} \tag{14.7.3}$$

Once these three cross-ratios have been computed, P can be reconstructed as the intersection of the planes ABP, BCP, CAP for any choice of three-dimensional coordinates for the points A, B, C, D, F.

We detail the construction of k_0. The other cross-ratios are obtained by permuting the points in the basis and their construction is omitted for the sake of conciseness. The diagram below shows the geometry of the reconstruction. The baseline $O'O''$ intersects the planes ABC, ABD, and ABF in three (unknown) points E_1, E_2, and E_3, and the visual ray $O'P$ intersects these three planes in Q, R, and S respectively.



As in the four-point case, we use the projective correspondence between the first image plane equipped with the a', b', c', e' basis, the plane ABC equipped with the A, B, C, E_1 basis, and the second image plane equipped with the a'', b'', c'', e'' basis, to reconstruct the image q'' of Q in the second image plane. Likewise, we reconstruct the images r'', s'' of the points R, S by exploiting planar projective correspondences. Since the four points Q, R, S, P project onto the four points q'', r'', s'', p'', the latter are collinear, and the cross-ratios of both four-tuples are the same.

Finally, intersecting the pencil ABC, ABD, ABF, ABP with the plane $O'AP$, we obtain:

$$k_0 = [AQ, AR, AS, AP] = [q'', r'', s'', p'']\,. \tag{14.7.4}$$

Similar constructions can be used to compute the cross-ratios k_1 and k_2.

Part V

MID-LEVEL VISION

Chapter 15

SEGMENTATION USING CLUSTERING METHODS

An attractive broad view of vision is that it is an inference problem: we have some measurements, and we wish to determine what caused them, using a mode. There are crucial features that distinguish vision from many other inference problems: firstly, there is an awful lot of data, and secondly, we don't know which of these data items come from objects — and so help with solving the inference problem — and which do not. For example, it is very difficult to tell whether a pixel lies on the dalmation in figure 15.1 simply by looking at the pixel. This problem can be addressed by working with a compact representation of the "interesting" image data that emphasizes the properties that make it "interesting". Obtaining this representation is known as **segmentation**.

It's hard to see that there could be a comprehensive theory of segmentation, not least because what is interesting and what is not depends on the application. There is certainly no comprehensive theory of segmentation at time of writing, and the term is used in different ways in different quarters. In this chapter we describe segmentation processes that have no probabilistic interpretation. In the following chapter, we deal with more complex probabilistic algorithms.

Segmentation is a broad term, covering a wide variety of problems and of techniques. We have collected a representative set of ideas in this chapter and in chapter **??**. These methods deal with different kinds of data set: some are intended for images, some are intended for video sequences and some are intended to be applied to **tokens** — placeholders that indicate the presence of an interesting pattern, say a spot or a dot or an edge point (figure 15.1). While superficially these methods may seem quite different, there is a strong similarity amongst them[1]. Each method attempts to obtain a compact representation of its data set using some form of model of similarity (in some cases, one has to look quite hard to spot the model).

One natural view of segmentation is that we are attempting to determine which components of a data set naturally "belong together". This is a problem known as **clustering**; there is a wide literature. Generally, we can cluster in two ways:

---

[1] Which is why they appear together!

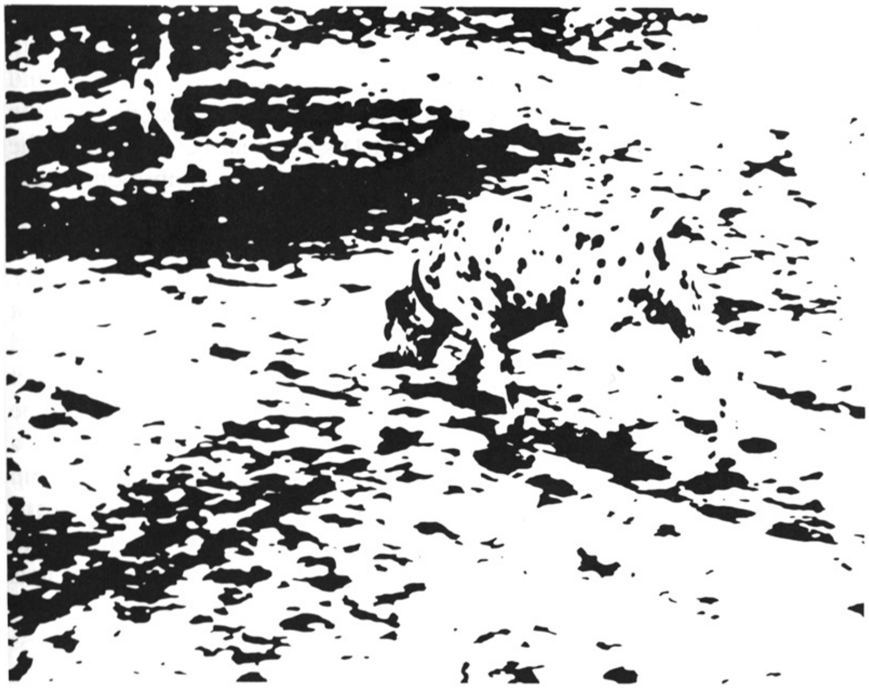


Figure 15.1. As the image of a dalmation on a shadowed background indicates, an important component of vision involves organising image information into meaningful assemblies. The human vision system seems to be able to do so surprisingly well. The blobs that form the dalmation appear to be assembled "because they form a dalmation," hardly a satisfactory explanation, and one that begs difficult computational questions. This process of organisation can be applied to many different kinds of input. *figure from Marr, Vision, page101, in the fervent hope that permission will be granted*

- **Partitioning:** here we have a large data set, and carve it up according to some notion of the association between items inside the set. We would like to decompose it into pieces that are "good" according to our model. For example, we might:
  - decompose an image into regions which have coherent colour and texture inside them;
  - take a video sequence and decompose it into **shots** — segments of video showing about the same stuff from about the same viewpoint;
  - decompose a video sequence into motion blobs, consisting of regions that have coherent colour, texture and motion.
- **Grouping:** here we have a set of distinct data items, and wish to collect sets of data items that "make sense" together according to our model. Effects like

occlusion mean that image components that belong to the same object are often separated. Examples of grouping include:

- collecting together tokens that, taken together, forming an interesting object (as in collecting the spots in figure 15.1);
- collecting together tokens that seem to be moving together .

15.1 Human vision: Grouping and Gestalt

Early psychophysics studied the extent to which a stimulus needed to be changed to obtain a change in response. For example, Webers' law attempts to capture the relationship between the intensity of a stimulus and its perceived brightness for very simple stimuli. The Gestalt school of psychologists rejected this approach, and emphasized grouping as an important part of understanding human vision. A common experience of segmentation is the way that an image can resolve itself into a **figure** — typically, the significant, important object — and a **ground** — the background on which the figure lies. However, as figure 15.2 illustrates, what is figure and what is ground can be profoundly ambiguous, meaning that a richer theory is required.

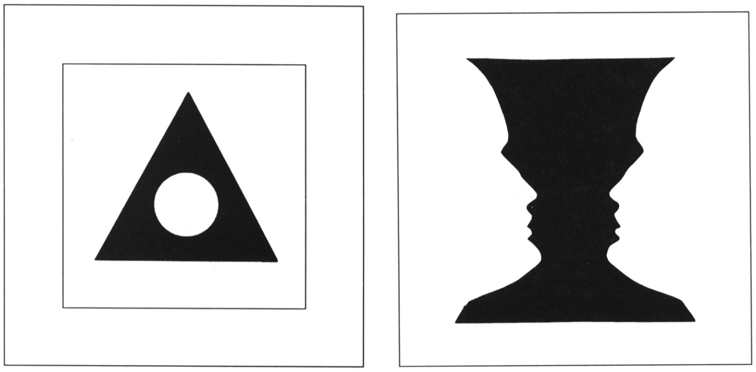


Figure 15.2. One view of segmentation is that it determines which component of the image forms the figure, and which the ground. The figure on the left illustrates one form of ambiguity that results from this view; the white circle can be seen as figure on the black triangular ground, or as ground where the figure is a black triangle with a circular whole in it — the ground is then a white square. On the right, another ambiguity: if the figure is black, then the image shows a vase, but if it is white, the image shows a pair of faces. *figure from Gordon, Theories of Visual Perception, page 65,66 in the fervent hope that permission will be granted*

The Gestalt school used the notion of a **gestalt** — a whole or a group — and of its **gestaltqualität** — the set of internal relationships that makes it a whole

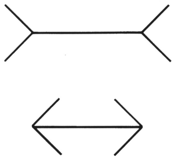


Figure 15.3. The famous Muller-Lyer illusion; the horizontal lines are in fact the same length, though that belonging to the upper figure looks longer. Clearly, this effect arises from some property of the relationships that form the whole (the *gestaltqualität*), rather than from properties of each separate segment. *figure from Gordon, Theories of Visual Perception, page 71 in the fervent hope that permission will be granted*

(e.g. figure 15.3) as central components in their ideas. Their work was characterised by attempts to write down a series of rules by which image elements would be associated together and interpreted as a group. There were also attempts to construct algorithms, which are of purely historical interest (see [**?**] for an introductory account that places their work in a broad context).

The Gestalt psychologists identified a series of factors, which they felt predisposed a set of elements to be grouped. There are a variety of factors, some of which postdate the main Gestalt movement:

- **Proximity:** tokens that are nearby tend to be grouped.
- **Similarity:** similar tokens tend to be grouped together.
- **Common fate:** tokens that have coherent motion tend to be grouped together.
- **Common region:** tokens that lie inside the same closed region tend to be grouped together.
- **Parallelism:** parallel curves or tokens tend to be grouped together.
- **Closure:** tokens or curves that tend to lead to closed curves tend to be grouped together.
- **Symmetry:** curves that lead to symmetric groups are grouped together.
- **Continuity:** tokens that lead to "continuous" — as in "joining up nicely", rather than in the formal sense — curves tend to be grouped.
- **Familiar Configuration:** tokens that, when grouped, lead to a familiar object, tend to be grouped together — familiar configuration can be seen as the reason that the tokens of figure 15.1 are all collected into a dalmation and a tree.

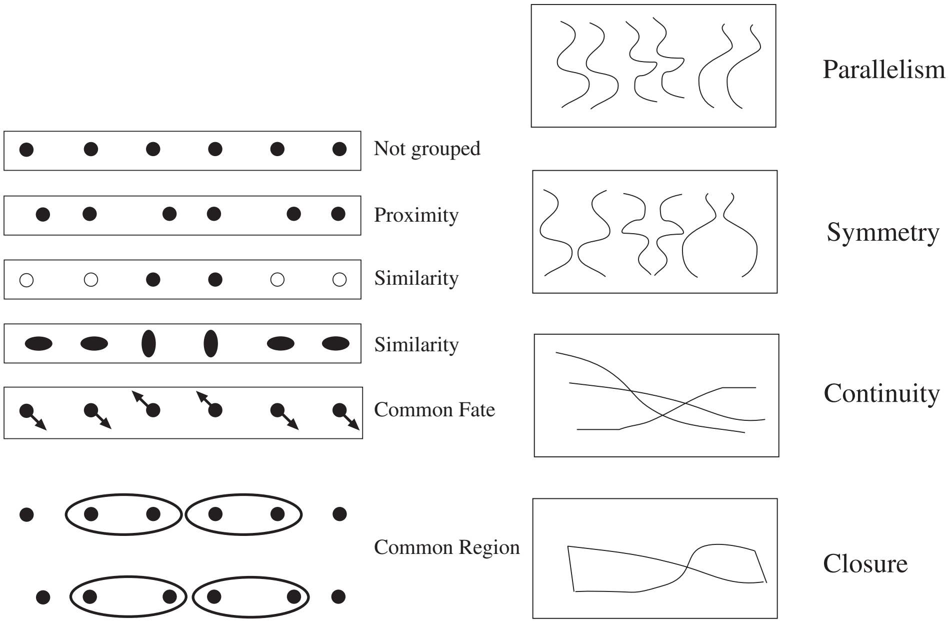


Figure 15.4. Examples of Gestalt factors that lead to grouping (which are described in greater detail in the text). *figure from Gordon, Theories of Visual Perception, page 67 in the fervent hope that permission will be granted*

These rules can function fairly well as explanations, but they are insufficiently crisp to be regarded as forming an algorithm. The Gestalt psychologists had serious difficulty with the details, such as when one rule applied and when another. It is very difficult to supply a satisfactory algorithm for using these rules — the Gestalt movement attempted to use an extremality principle.

Familiar configuration is a particular problem. The key issue is to understand just *what* familiar configuration applies in a problem, and how it is selected. For example, look at figure 15.1; one might argue that the blobs are grouped because they yield a dog. The difficulty with this view is explaining how this occurred — where did the hypothesis that a dog is present come from? a search through all views of all objects is one explanation, but one must then explain how this search is organised — do we check *every view* of *every* dog with *every* pattern of spots? how can this be done efficiently?

The Gestalt rules do offer some insight, because they offer some explanation for what happens in various examples. These explanations seem to be sensible, because they suggest that the rules help solve problems posed by visual effects that arise commonly in the real world — that is, they are **ecologically valid**. For example, continuity may represent a solution to problems posed by occlusion — sections of the contour of an occluded object could be joined up by continuity (see figures **??**

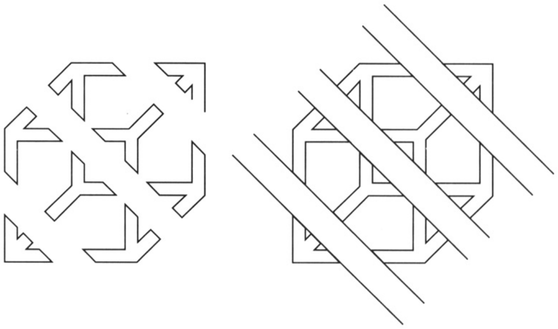


Figure 15.5. Occlusion appears to be an important cue in grouping. With some effort, the pattern on the left can be seen as a cube, whereas the pattern on the right is clearly and immediately a cube. The visual system appears to be helped by evidence that separated tokens are separated for a reason, rather than just scattered. *figure from Gordon, Theories of Visual Perception, page 87 in the fervent hope that permission will be granted*

and 15.5).

This tendency to prefer interpretations that are explained by occlusion leads to interesting effects. One is the **illusory contour**, illustrated in figure 15.6. Here a set of tokens suggests the presence of an object most of whose contour has no contrast. The tokens appear to be grouped together because they provide a cue to the presence of an occluding object, which is so strongly suggested by these tokens that one could fill in the no-contrast regions of contour.

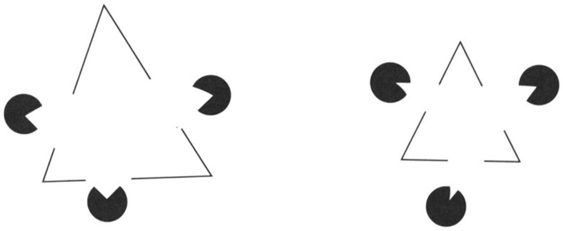


Figure 15.6. The tokens in these images suggest the presence of occluding triangles, whose boundaries don't contrast with much of the image, except at their vertices. Notice that one has a clear impression of the position of the entire contour of the occluding figures. These contours are known as *illusory contours. figure from Marr, Vision, page51, in the fervent hope that permission will be granted*

This ecological argument has some force, because it is possible to interpret most grouping factors using it. Common fate can be seen as a consequence of the fact that components of objects tend to move together. Equally, symmetry is a useful grouping cue because there are a lot of real objects that have symmetric or close to symmetric contours. Essentially, the ecological argument says that tokens are grouped because doing so produces representations that are helpful for the visual world that people encounter. The ecological argument has an appealing, though vague, statistical flavour. From our perspective, Gestalt factors provide interesting hints, but should be seen as the *consequences* of a larger grouping process, rather than the process itself.

15.2 Applications: Shot Boundary Detection, Background Subtraction and Skin Finding

Simple segmentation algorithms are often very useful in significant applications. Generally, simple algorithms work best when it is very easy to tell what a “useful” decomposition is. Three important cases are **background subtraction** — where anything that doesn’t look like a known background is interesting — **shot boundary detection** — where substantial changes in a video are interesting — and **skin finding** — where pixels that look like human skin are interesting.

15.2.1 Background Subtraction

In many applications, objects appear on a background which is very largely stable. The standard example is detecting parts on a conveyor belt. Another example is counting motor cars in an overhead view of a road — the road itself is pretty stable in appearance. Another, less obvious, example is in human computer interaction. Quite commonly, a camera is fixed (say, on top of a monitor) and views a room. Pretty much anything in the view that doesn’t look like the room is interesting.

In these kinds of applications, a useful segmentation can often be obtained by subtracting an estimate of the appearance of the background from the image, and looking for large absolute values in the result. The main issue is obtaining a good estimate of the background. One method is simply to take a picture. This approach works rather poorly, because the background typically changes slowly over time. For example, the road may get more shiny as it rains and less when the weather dries up; people may move books and furniture around in the room, etc.

An alternative which usually works quite well is to estimate the value of background pixels using a **moving average**. In this approach, we estimate the value of a particular background pixel as a weighted average of the previous values. Typically, pixels in the very distant past should be weighted at zero, and the weights increase smoothly. Ideally, the moving average should track the changes in the background, meaning that if the weather changes very quickly (or the book mover is frenetic) relatively few pixels should have non-zero weights, and if changes are slow, the number of past pixels with non-zero weights should increase. This yields algorithm 1 For those who have read the filters chapter, this is a filter that smooths a function of time, and we would like it to suppress frequencies that are larger than the typical frequency of change in the background and pass those that are at or below that frequency. As figures 15.7 and 15.8 indicate, the approach can be quite

```
Form a background estimate $\mathcal{B}^{(0)}$.
At each frame $\mathcal{F}$

    Update the background estimate, typically by
    forming $\mathcal{B}^{(n+1)} = \frac{w_a\mathcal{F}+\sum_i w_i\mathcal{B}^{(n-i)}}{w_c}$
    for a choice of weights $w_a$, $w_i$ and $w_c$.

    Subtract the background estimate from the
    frame, and report the value of each pixel where
    the magnitude of the difference is greater than some
    threshold.

end
```

Algorithm 15.1: *Background Subtraction*

successful.

Missing
Figure

Figure 15.7. Moving average results for human segmentation

15.2.2 Shot Boundary Detection

Long sequences of video are composed of **shots** — much shorter subsequences that show largely the same objects. These shots are typically the product of the editing process. There is seldom any record of where the boundaries between shots fall. It is helpful to represent a video as a collection of shots; each shot can then be represented with a **key frame**. This representation can be used to search for videos or to encapsulate their content for a user to browse a video or a set of videos.

Finding the boundaries of these shots automatically — **shot boundary detection** — is an important practical application of simple segmentation algorithms.

Missing
Figure

Figure 15.8. Moving average results for car segmentation

A shot boundary detection algorithm must find frames in the video that are "significantly" different from the previous frame. Our test of significance must take account of the fact that within a given shot both objects and the background can move around in the field of view. Typically, this test takes the form of a distance; if the distance is larger than a threshold, a shot boundary is declared (algorithm 2).

```
For each frame in an image sequence

    Compute a distance between this frame and the
    previous frame

    If the distance is larger than some threshold,

      classify the frame as a shot boundary.

end
```

Algorithm 15.2: *Shot boundary detection using interframe differences*

There are a variety of standard techniques for computing a distance:

- **Frame differencing** algorithms take pixel-by-pixel differences between each two frames in a sequence, and sum the squares of the differences. These algorithms are unpopular, because they are slow — there are many differences — and because they tend to find many shots when the camera is shaking.
- **Histogram based** algorithms compute colour histograms for each frame, and compute a distance between the histograms. A difference in colour histograms is a sensible measure to use, because it is insensitive to the spatial arrangement of colours in the frame — for example, small camera jitters will not affect the histogram.

Missing
Figure

Figure 15.9. Shot boundary detection results.

- **Block comparison** algorithms compare frames by cutting them into a grid of boxes, and comparing the boxes. This is to avoid the difficulty with colour histograms, where (for example) a red object disappearing off-screen in the bottom left corner is equivalent to a red object appearing on screen from the top edge. Typically, these block comparison algorithms compute an inter-frame distance that is a composite — taking the maximum is one natural strategy — of inter-block distances, computed using the methods above.

- **Edge differencing** algorithms compute edge maps for each frame, and then compare these edge maps. Typically, the comparison is obtained by counting the number of potentially corresponding edges (nearby, similar orientation, etc.) in the next frame. If there are few potentially corresponding edges, there is a shot boundary. A distance can be obtained by transforming the number of corresponding edges.

These are relatively *ad hoc* methods, but are often sufficient to solve the problem at hand.

15.2.3 Finding Skin Using Image Colour

It is often very useful to be able to find human skin in pictures. For example, gesture-based user interfaces usually need to know where the face and hands of the current user are. Similarly, if we were searching for pictures of people, a natural thing to do is to look for faces. Human skin has a surprisingly limited range of hues and is not deeply saturated. The colour largely results from the effects of blood and melanin (which contribute respectively red and yellow/brown hues). Skin cannot be modelled well with a BRDF, because its appearance is affected by scattering from quite deep below the surface and by oil and sweat films on the surface. These effects make skin extremely hard to render convincingly; [**?**] has the best account. For skin detection, we can safely ignore this reservation and obtain a reflectance with a substantial non-Lambertian component. Skin typically has bright areas due to specular reflection on oil and sweat films. These specularities take the illumination color, which varies slightly from image to image, so that some skin regions appear as blueish or greenish off-white. Finally, skin has little texture at a coarse scale.

In practice, skin can be found quite effectively by looking for pixels with colours in a given range and where there is little coarse-scale texture. There are many methods, and the details vary somewhat from method to method. We sketch a method due to Forsyth and Fleck [**?**]; we discuss another method in greater detail in section **??**. Forsyth and Fleck transform image RGB values to an opponent colour representation that uses intensity and two intensity independent opponent channels. This representation is smoothed. Each pixel on the resulting intensity plane is tested for texture intensity. Texture intensity is estimated by forming the median intensity of a neighbourhood around each pixel; subtracting this median from the intensity of each pixel in the neighbourhood; and taking the median of the absolute value of the result.

Corresponding pixels on the intensity independent opponent channels are tested to determine whether their colour lies within a given range. Pixels whose colour lies in this range and whose texture intensity is below some threshold are marked as being “probably skin”. Because this process tends not to mark specular patches on skin, all pixels with many neighbours that are “probably skin” are marked as “probably skin”, too. The resulting collection of pixels is tested against a second range of colours; all pixels that pass both tests are regarded as skin pixels.

The success of these methods seems to be due to cultural practices. Skin changes colour as it is viewed under different coloured lights, just like any other material, but people seem to choose lighting and to adjust pictures to reduce these effects, probably because pictures of skin where the hue is too green or too blue are very disturbing (subjects tend to look sick or dead, respectively).

15.3 Image Segmentation by Clustering Pixels

Clustering is a process whereby a data set is replaced by **clusters**, which are collections of data points that “belong together”. It is natural to think of image segmentation as clustering; we would like to represent an image in terms of clusters of pixels that “belong together”. The specific criterion to be used depends on the application. Pixels may belong together because they have the same colour and/or they have the same texture and/or they are nearby, etc.

15.3.1 Simple Clustering Methods

There are two natural algorithms for clustering. In **divisive clustering**, the entire data set is regarded as a cluster, and then clusters are recursively split to yield a good clustering (algorithm 4). In **agglomerative clustering**, each data item is regarded as a cluster and clusters are recursively merged to yield a good clustering (algorithm 3).

There are two major issues in thinking about clustering:

- *what is a good inter-cluster distance?* Agglomerative clustering uses an inter-

Missing
Figure

Figure 15.10. Marking skin pixels is useful for, for example, finding faces or hands. The figures on the top show a series of images of faces; below, we show the output of the skin marking process described in the text. While colour and texture are not an exact test for skin, much of the background is masked and faces generally appear. Notice that this approach works reasonably well for a range of skin colours.

```
Make each point a separate cluster

Until the clustering is satisfactory

    Merge the two clusters with the
    smallest inter-cluster distance

end
```

Algorithm 15.3: *Agglomerative clustering, or clustering by merging*

cluster distance to fuse "nearby" clusters; divisive clustering uses it to split insufficiently "coherent" clusters. Even if a natural distance between data points is available (which may not be the case for vision problems), there is no canonical inter-cluster distance. Generally, one chooses a distance that seems appropriate for the data set. For example, one might choose the distance between the closest elements as the inter-cluster distance — this tends to yield extended clusters (statisticians call this method **single-link clustering**). Another natural choice is the maximum distance between an element of the first cluster and one of the second — this tends to yield "rounded" clusters (statisticians call this method **complete-link clustering**). Finally, one could use an average of distances between elements in the clusters — this will also tend to yield "rounded" clusters (statisticians call this method **group**

```
Construct a single cluster containing all points

Until the clustering is satisfactory

    Split the cluster that yields the two
    components with the largest inter-cluster distance

end
```

Algorithm 15.4: *Divisive clustering, or clustering by splitting*

average clustering).

- *and how many clusters are there?* This is an intrinsically difficult task if there is no model for the process that generated the clusters. The algorithms we have described generate a hierarchy of clusters. Usually, this hierarchy is displayed to a user in the form of a **dendrogram** — a representation of the structure of the hierarchy of clusters that displays inter-cluster distances — and an appropriate choice of clusters is made from the dendrogram (see the example in figure 15.11).

15.3.2 Segmentation Using Simple Clustering Methods

It is relatively easy to take a clustering method and build an image segmenter from it. Much of the literature on image segmentation consists of papers that are, in essence, papers about clustering (though this isn't always acknowledged). The distance used depends entirely on the application, but measures of colour difference and of texture are commonly used as clustering distances. It is often desirable to have clusters that are "blobby"; this can be achieved by using difference in position in the clustering distance.

The main difficulty in using either agglomerative or divisive clustering methods directly is that there are an awful lot of pixels in an image. There is no reasonable prospect of examining a dendrogram, because the quantity of data means that it will be too big. Furthermore, the mechanism is suspect; we don't really want to look at a dendrogram for each image, but would rather have the segmenter produce useful regions for an application on a long sequence of images without any help. In practice, this means that the segmenters decide when to stop splitting or merging by using a set of threshold tests — for example, an agglomerative segmenter may stop merging when the distance between clusters is sufficiently low, or when the number of clusters reaches some value. The choice of thresholds is usually made by observing the behaviour of the segmenter on a variety of images, and choosing the best setting. The technique has largely fallen into disuse except in

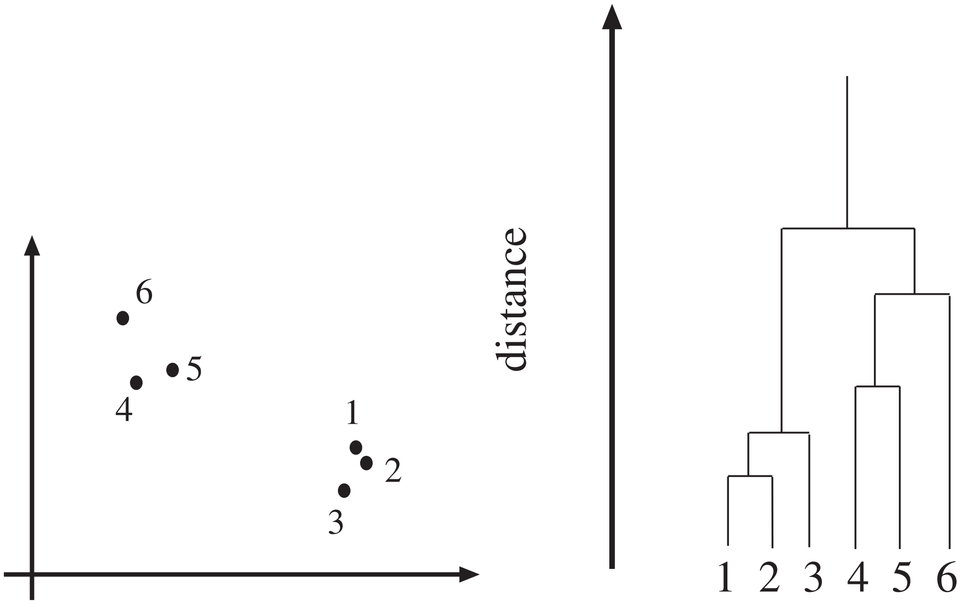


Figure 15.11. Left, a data set; right, a dendrogram obtained by agglomerative clustering using single link clustering. If one selects a particular value of distance, then a horizontal line at that distance will split the dendrogram into clusters. This representation makes it possible to guess how many clusters there are, and to get some insight into how good the clusters are.

specialised applications, because in most cases it is very difficult to predict the future performance of the segmenter tuned in this way.

Another difficulty created by the number of pixels is that it is impractical to look for the best split of a cluster (for a divisive method) or the best merge (for an agglomerative method). The variety of tricks that have been adopted to address this problem is far too large to survey here, but we can give an outline of the main strategies.

Divisive methods are usually modified by using some form of summary of a cluster to suggest a good split. A natural summary to use is a histogram of pixel colours (or grey levels). In one of the earliest segmentation algorithms, due to Ohlander [**?**], regions are split by identifying a peak in one of nine feature histograms (these are colour coordinates of the pixel in each of three different colour spaces) and attempting to separate that peak from the histogram. Of course, textured regions need to be masked to avoid splitting texture components apart. Figures 15.13 and 15.14 illustrate this segmenter.

Agglomerative methods also need to be modified. There are three main issues:

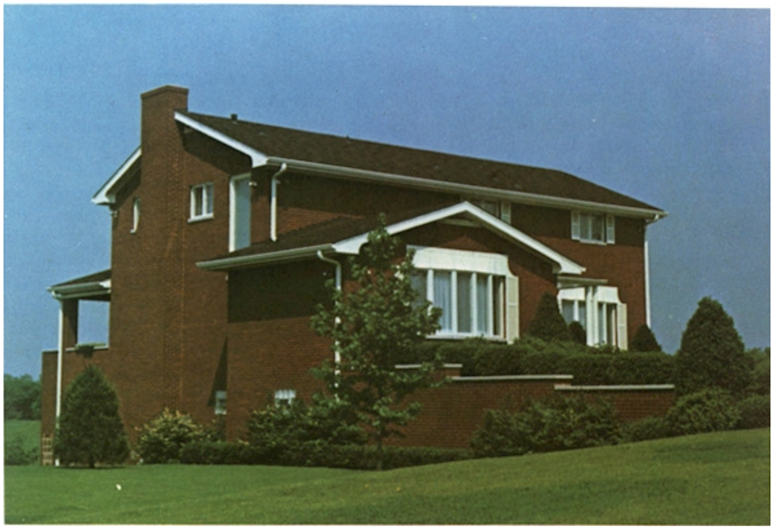


Figure 15.12. We illustrate an early segmenter that uses a divisive clustering algorithm, due to [**?**] (circa 1975) using this figure of a house, which is segmented into the hierarchy of regions indicated in figure 15.13.

- Firstly, given two clusters containing large numbers of pixels, it is expensive to find the average distance or the minimum distance between elements of the clusters; alternatives include the distance between centers of gravity.

- Secondly, it is usual to try and merge only clusters with shared boundaries (this can be accounted for by attaching a term to the distance function that is zero for neighbouring pixels and infinite for all others). This approach avoids clustering together regions that are widely separated (we probably don't wish to represent the US flag as three clusters, one red, one white and one blue).

- Finally, it can be useful to merge regions simply by scanning the image and merging all pairs whose distance falls below a threshold, rather than searching for the closest pair. This strategy means the dendrogram is meaningless, but the dendrogram is so seldom used this doesn't usually matter.

15.3.3 Clustering and Segmentation by K-means

Simple clustering methods use greedy interactions with existing clusters to come up with a good overall representation. For example, in agglomerative clustering we repeatedly make the best available merge. However, the methods are not explicit about the objective function that the methods are attempting to optimize. An alternative approach is to write down an objective function that expresses how good a representation is, and then build an algorithm for obtaining the best representation.

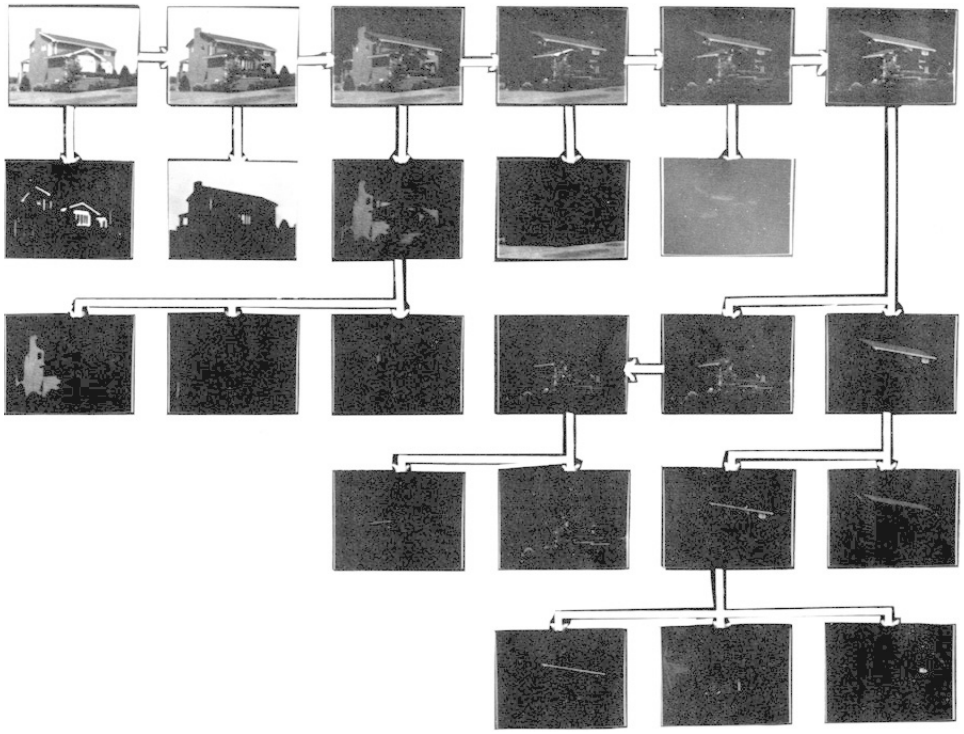


Figure 15.13. The hierarchy of regions obtained from figure 15.12, by a divisive clustering algorithm. A typical histogram is shown in figure 15.14. The segmentation process is stopped when regions satisfy an internal coherence test, defined by a collection of fixed thresholds.

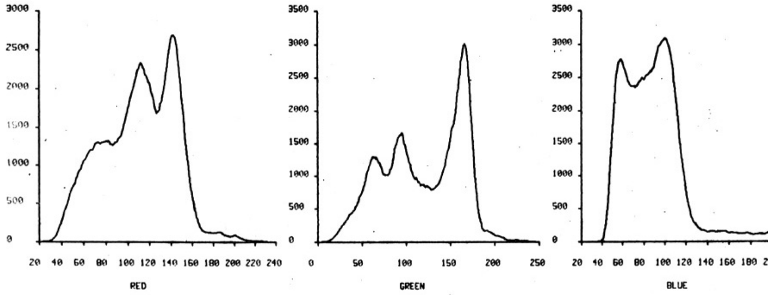


Figure 15.14. A histogram encountered while segmenting figure 15.12 into the hierarchy of figure 15.13 using the divisive clustering algorithm of [**?**].

A natural objective function can be obtained by assuming that we know there are k clusters, where k is known. Each cluster is assumed to have a center; we write the center of the i'th cluster as $\boldsymbol{c}_i$. The j'th element to be clustered is described by a feature vector $\boldsymbol{x}_j$. For example, if we were segmenting scattered points, then $\boldsymbol{x}$ would be the coordinates of the points; if we were segmenting an intensity image, $\boldsymbol{x}$ might be the intensity at a pixel.

We now assume that elements are close to the center of their cluster, yielding the objective function

$$\Phi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{`th cluster}} (\boldsymbol{x}_j - \boldsymbol{c}_i)^T (\boldsymbol{x}_j - \boldsymbol{c}_i) \right\}$$

Notice that if the allocation of points to clusters is known, it is easy to compute the best center for each cluster. However, there are far too many possible allocations of points to clusters to search this space for a minimum. Instead, we define an algorithm which iterates through two activities:

- Assume the cluster centers are known, and allocate each point to the closest cluster center.
- Assume the allocation is known, and choose a new set of cluster centers. Each center is the mean of the points allocated to that cluster.

We then choose a start point by randomly choosing cluster centers, and then iterate these stages alternately. This process will eventually converge to a local minimum of the objective function (why?). It is not guaranteed to converge to the global minimum of the objective function, however. It is also not guaranteed to produce k clusters, unless we modify the allocation phase to ensure that each cluster has some non-zero number of points. This algorithm is usually referred to as **k-means**. It is possible to search for an appropriate number of clusters by applying k-means for different values of k, and comparing the results; we defer a discussion of this issue until section 17.3.

One difficulty with using this approach for segmenting images is that segments are not connected and can be scattered very widely (figures 15.15 and 15.16). This effect can be reduced by using pixel coordinates as features, an approach that tends to result in large regions being broken up (figure 15.17).

15.4 Segmentation by Graph-Theoretic Clustering

Clustering can be seen as a problem of cutting graphs into "good" pieces. In effect, we associate each data item with a vertex in a weighted graph, where the weights on the edges between elements are large if the elements are "similar" and small if they are not. We then attempt to cut the graph into connected components with relatively large interior weights — which correspond to clusters — by cutting edges with relatively low weights. This view leads to a series of different, quite successful, segmentation algorithms.

```
Choose $k$ data points to act as cluster centers

Until the cluster centers are unchanged

    Allocate each data point to cluster whose center is nearest

    Now ensure that every cluster has at least
    one data point; possible techniques for doing this include .
    supplying empty clusters with a point chosen at random from
    points far from their cluster center.

    Replace the cluster centers with the mean of the elements
    in their clusters.

end
```

Algorithm 15.5: *Clustering by K-Means*

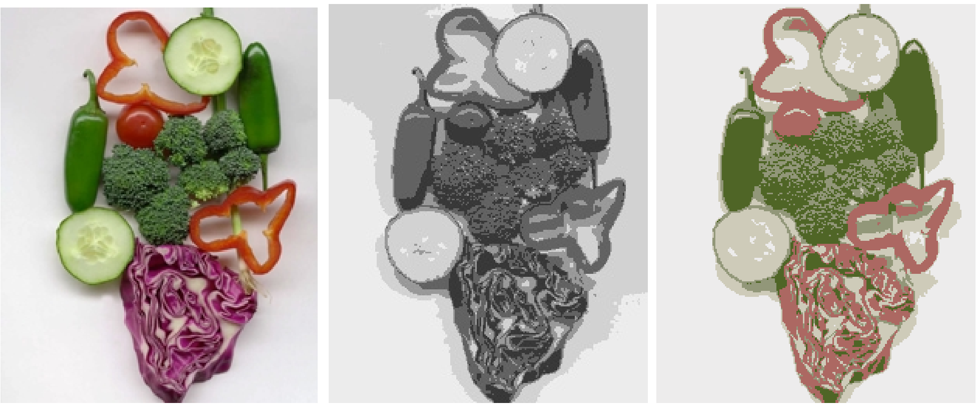


Figure 15.15. On the left, an image of mixed vegetables, which is segmented using k-means to produce the images at center and on the right. We have replaced each pixel with the mean value of its cluster; the result is somewhat like an adaptive requantization, as one would expect. In the center, a segmentation obtained using only the intensity information. At the right, a segmentation obtained using colour information. Each segmentation assumes five clusters.

15.4.1 Basic Graphs

We review terminology here very briefly, as it's quite easy to forget.

- A **graph** is a set of vertices V and edges E which connect various pairs of

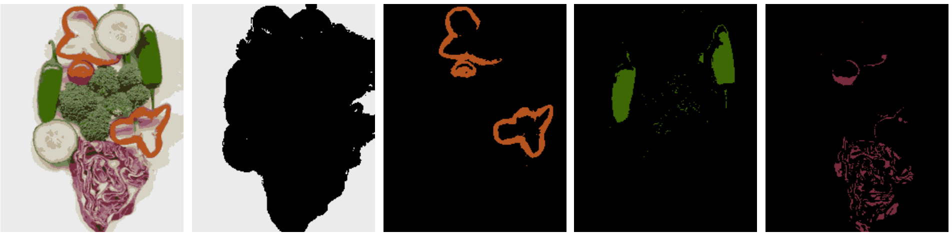


Figure 15.16. Here we show the image of vegetables segmented with k-means, assuming a set of 11 components. The top left figure shows all segments shown together, with the mean value in place of the original image values. The other figures show four of the segments. Note that this approach leads to a set of segments that are not necessarily connected. For this image, some segments are actually quite closely associated with objects but one segment may represent many objects (the peppers); others are largely meaningless. The absence of a texture measure creates serious difficulties, as the many different segments resulting from the slice of red cabbage indicate.

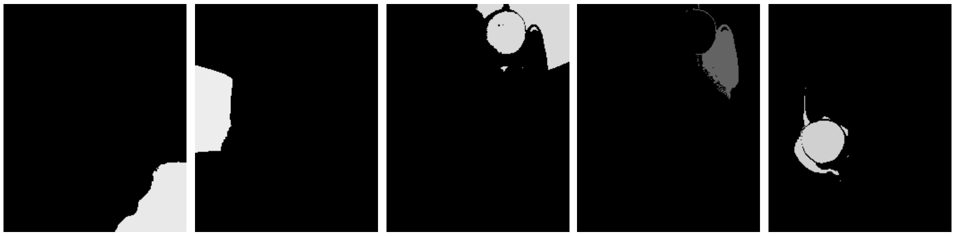


Figure 15.17. Five of the segments obtained by segmenting the image of vegetables with a k-means segmenter that uses position as part of the feature vector describing a pixel, now using 20 segments rather than 11. Note that the large background regions that should be coherent has been broken up because points got too far from the center. The individual peppers are now better separated, but the red cabbage is still broken up because there is no texture measure.

vertices. A graph can be written $G = \{V, E\}$. Each edge can be represented by a pair of vertices, that is $E \subset V \times V$. Graphs are often drawn as a set of points with curves connecting the points.

- A **directed graph** is one in which edges (a, b) and (b, a) are distinct; such a graph is drawn with arrowheads indicating which direction is intended.
- An **undirected graph** is one in which no distinction is drawn between edges (a, b) and (b, a).
- A **weighted graph** is one in which a weight is associated with each edge.
- A **self-loop** is an edge that has the same vertex at each end; self-loops don't occur in practice in our applications.

- Two vertices are said to be **connected** if there is a sequence of edges starting at the one and ending at the other; if the graph is directed, then the arrows in this sequence must point the right way.
- A **connected graph** is one where every pair of vertices is connected.
- Every graph consists of a disjoint set of **connected components**, that is $G = \{V_1 \cup V_2 \ldots V_n, E_1 \cup E_2 \ldots E_n\}$, where $\{V_i, E_i\}$ are all connected graphs and there is no edge in E that connects an element of V_i with one of V_j for $i \neq j$.

15.4.2 The Overall Approach

It is useful to understand that a weighted graph can be represented by a square matrix (figure 15.18). There is a row and a column for each vertex. The i, j'th element of the matrix represents the weight on the edge from vertex i to vertex j; for an undirected graph, we use a symmetric matrix and place half the weight in each of the i, j'th and j, i'th element.

The application of graphs to clustering is this: take each element of the collection to be clustered, and associate it with a vertex on a graph. Now construct an edge from every element to every other, and associate with this edge a weight representing the extent to which the elements are similar. Now cut edges in the graph to form a "good" set of connected components. Each of these will be a cluster. For example, figure 15.19 shows a set of well separated points and the weight matrix (i.e. undirected weighted graph, just drawn differently) that results from a particular similarity measure; a desirable algorithm would notice that this matrix looks a lot like a block diagonal matrix — because intercluster similarities are strong and intracluster similarities are weak — and split it into two matrices, each of which is a block. The issues to study are the criteria that lead to good connected components and the algorithms for forming these connected components.

15.4.3 Affinity Measures

When we viewed segmentation as simple clustering, we needed to supply some measure of how similar clusters were. The current model of segmentation simply requires a weight to place on each edge of the graph; these weights are usually called **affinity measures** in the literature. Clearly, the affinity measure depends on the problem at hand. The weight of an arc connecting similar nodes should be large, and the weight on an arc connecting very different nodes should be small. It is fairly easy to come up with affinity measures with these properties for a variety of important cases, and we can construct an affinity function for a combination of cues by forming a product of powers of these affinity functions. You should be aware that other choices of affinity function are possible; there is no particular reason to believe that a canonical choice exists.

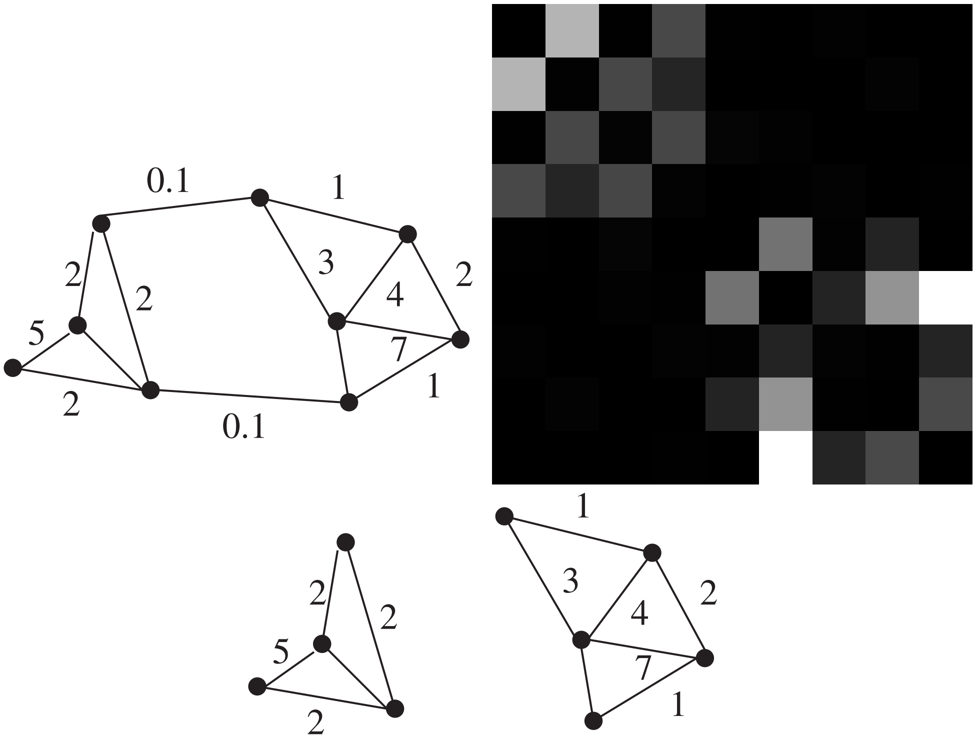


Figure 15.18. On the **top left**, a drawing of an undirected weighted graph; on the **top right**, the weight matrix associated with that graph. Larger values are lighter. By associating the vertices with rows (and columns) in a different order, the matrix can be shuffled. We have chosen the ordering to show the matrix in a form that emphasizes the fact that it is very largely block-diagonal. The figure on the **bottom** shows a cut of that graph that decomposes the graph into two tightly linked components. This cut decomposes the graph's matrix into the two main blocks on the diagonal.

Affinity by Distance

Affinity should go down quite sharply with distance, once the distance is over some threshold. One appropriate expression has the form

$$\mathrm{aff}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{-\big((\boldsymbol{x}-\boldsymbol{y})^t(\boldsymbol{x}-\boldsymbol{y})/2\sigma_d^2\big)\right\}$$

where σ_d is a parameter which will be large if quite distant points should be grouped and small if only very nearby points should be grouped (this is the expression used for figure 15.19).

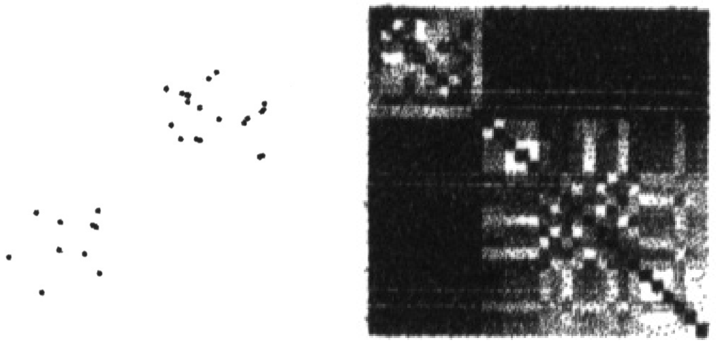


Figure 15.19. On the left, a set of points on the plane. On the right, the affinity matrix for these points computed using a decaying exponential in distance (section 15.4.3), where large values are light and small values are dark. Notice the near block diagonal structure of this matrix; there are two off-diagonal blocks that contain terms that are very close to zero. The blocks correspond to links internal to the two obvious clusters, and the off diagonal blocks correspond to links between these clusters. *figure from Perona and Freeman, A factorization approach to grouping, page 2 figure from Perona and Freeman, A factorization approach to grouping, page 4*

Affinity by Intensity

Affinity should be large for similar intensities, and smaller as the difference increases. Again, an exponential form suggests itself, and we can use:

$$\mathrm{aff}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{-\left((I(\boldsymbol{x}) - I(\boldsymbol{y}))^t (I(\boldsymbol{x}) - I(\boldsymbol{y}))/2\sigma_I^2\right)\right\}$$

Affinity by Colour

We need a colour metric to construct a meaningful colour affinity function. It's a good idea to use a uniform colour space, and a bad idea to use RGB space, — for reasons that should be obvious, otherwise, reread section **??** — and an appropriate expression has the form

$$\mathrm{aff}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{-\left(\mathrm{dist}(\boldsymbol{c}(\boldsymbol{x}), \boldsymbol{c}(\boldsymbol{y}))^2/2\sigma_c^2\right)\right\}$$

where $\boldsymbol{c}_i$ is the colour at pixel i.

Affinity by Texture

The affinity should be large for similar textures and smaller as the difference increases. We adopt a collection of filters $f_1, \ldots, f_n$, and describe textures by the outputs of these filters, which should span a range of scales and orientations. Now for most textures, the filter outputs will not be the same at each point in the texture — think of a chessboard — but a histogram of the filter outputs constructed over a reasonably sized neighbourhood will be well behaved. For example, in the case of an infinite chessboard, if we take a histogram of filter outputs over a region that covers a few squares, we can expect this histogram to be the same wherever the region falls.

This suggests a process where we firstly establish a local scale at each point — perhaps by looking at energy in coarse scale filters, or using some other method — and then compute a histogram of filter outputs over a region determined by that scale — perhaps a circular region centered on the point in question. We then write $\boldsymbol{h}$ for this histogram, and use an exponential form:

$$\mathrm{aff}(\boldsymbol{x},\boldsymbol{y}) = \exp\left\{-\left((\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{f}(\boldsymbol{y}))^t(\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{f}(\boldsymbol{y}))/2\sigma_I^2\right)\right\}$$

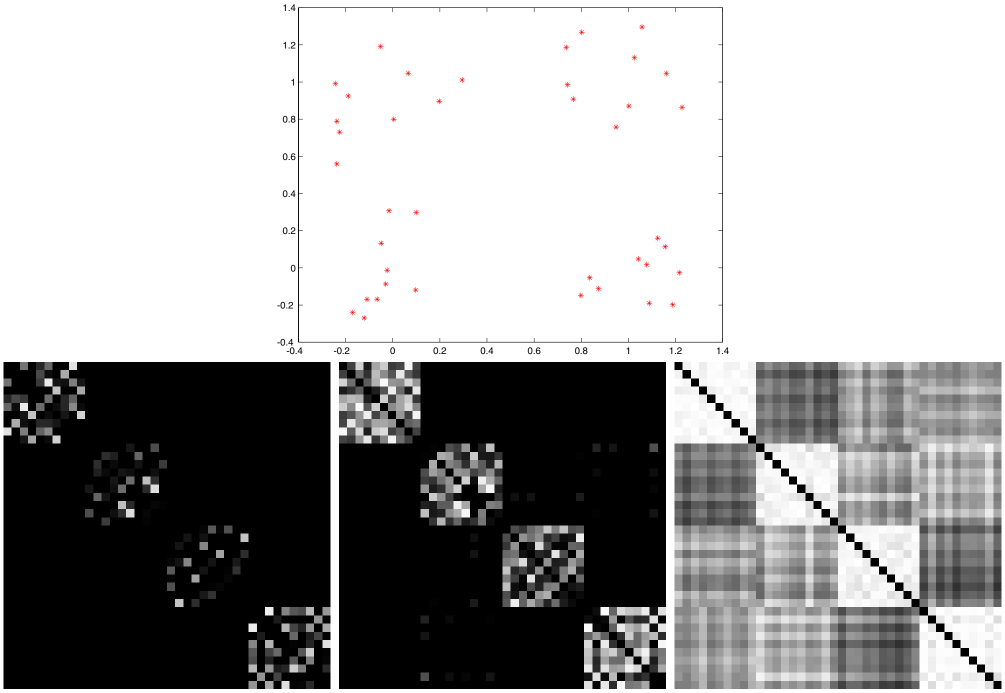


Figure 15.20. The choice of scale for the affinity affects the affinity matrix. The top row shows a dataset, which consists of four groups of 10 points drawn from a rotationally symmetric normal distribution with four different means. The standard deviation in each direction for these points is 0.2. In the second row, affinity matrices computed for this dataset using different values of σ_d. On the **left**, $\sigma_d = 0.1$, in the **center** $\sigma_d = 0.2$ and on the **right**, $\sigma_d = 1$. For the finest scale, the affinity between all points is rather small; for the next scale, there are four clear blocks in the affinity matrix; and for the coarsest scale, the number of blocks is less obvious.

Affinity by Motion

In the case of motion, the nodes of the graph are going to represent a pixel in a particular image in the sequence. It is difficult to estimate the motion at a particular pixel accurately; instead, it makes sense to construct a distribution over the possible motions. The quality of motion estimate available depends on what the neighbourhood of the pixel looks like. For example, if the pixel lies on an edge, this motion component parallel to the edge is going to be uncertain but the component perpendicular to the edge is going to be quite well measured. One way to obtain a reasonable estimate of the probability distribution is to compare a translated version of the neighbourhood with the next image; if the two are similar, then the probability of this motion should be relatively high. If we define a similarity measure for an image motion $\boldsymbol{v}$ at a pixel $\boldsymbol{x}$ to be

$$S(\boldsymbol{v}, \boldsymbol{x}; \sigma_d) = \exp\left(-\frac{1}{2\sigma_d^2} \sum_{\boldsymbol{u} \in \text{neighbourhood}} \left\{ I_t(\boldsymbol{x}+\boldsymbol{u}+\boldsymbol{v}) - I_{t+1}(\boldsymbol{x}+\boldsymbol{u}) \right\}^2 \right)$$

we have a measure that will be near one for a good value of the motion and near zero for a poor one. This can be massaged into a probability distribution by ensuring that it somes to one, so we have

$$P(\boldsymbol{v}, \boldsymbol{x}; \sigma_d) = \frac{S_i(\boldsymbol{v}, \boldsymbol{x}; \sigma_d)}{\sum_{\boldsymbol{v}} S_i(\boldsymbol{v}, \boldsymbol{x}; \sigma_d)}$$

Now we need to obtain an affinity measure from this. The arcs on the graph will connect pixels that are "nearby" in space and in time. For each pair of pixels, the affinity should be high if the motion pattern around the pixels could look similar, and low otherwise. This suggests using a correlation measure for the affinity

$$\text{aff}(\boldsymbol{x}, \boldsymbol{y}; \sigma_d, \sigma_m) = \exp\left(-\frac{1}{2\sigma_m^2} \left\{ 1 - \sum_{\boldsymbol{v}} P(\boldsymbol{v}, \boldsymbol{x}; \sigma_d) P(\boldsymbol{v}, \boldsymbol{x}; \sigma_d) \right\} \right)$$

15.4.4 Eigenvectors and Segmentation

In the first instance, assume that there are k elements and k clusters. We can represent a cluster by a vector with k components. We will allow elements to be associated with clusters using some continuous weight — we need to be a bit vague about the semantics of these weights, but the intention is that if a component in a particular vector has a small value, then it is weakly associated with the cluster, and if it has a large value, then it is strongly associated with a cluster.

Extracting a Single Good Cluster

A good cluster is one where elements that are strongly associated with the cluster also have large values in the affinity matrix. Write the matrix representing the element affinities as $\mathcal{A}$, and the vector of weights as $\boldsymbol{w}$ In particular, we can construct an objective function

$$\boldsymbol{w}^T \mathcal{A} \boldsymbol{w}$$

This is a sum of terms of the form

$$\begin{aligned}\{\text{association of element } i \text{ with cluster}\} &\times \\ \{\text{affinity between } i \text{ and } j\} &\times \\ \{\text{association of element } j \text{ with cluster}\}&\end{aligned}$$

We can obtain a cluster by choosing a set of association weights that maximise this objective function. The objective function is useless on its own, because scaling $\boldsymbol{w}$ by λ scales the total association by λ^2. However, we can normalise the weights by requiring that $\boldsymbol{w}^T \boldsymbol{w} = 1$.

This suggests maximising $\boldsymbol{w}^T \mathcal{A} \boldsymbol{w}$ subject to $\boldsymbol{w}^T \boldsymbol{w} = 1$. The Lagrangian is

$$\boldsymbol{w}^T \mathcal{A} \boldsymbol{w} + \lambda \left(\boldsymbol{w}^T \boldsymbol{w} - 1 \right)$$

and differentiation and dropping a factor of two yields

$$\mathcal{A} \boldsymbol{w} = \lambda \boldsymbol{w}$$

meaning that $\boldsymbol{w}$ is an eigenvector of $\mathcal{A}$. This means that we could form a cluster by obtaining the eigenvector with the largest eigenvalue — the cluster weights are the elements of the eigenvector. For problems where reasonable clusters are apparent, we expect that these cluster weights are large for some elements — which belong to the cluster — and nearly zero for others — which do not. In fact, we can get the weights for other clusters from other eigenvectors of $\mathcal{A}$ as well.

Extracting Weights for a Set of Clusters

In the kind of problems we expect to encounter, there are strong association weights between relatively few pairs of elements. For example, if each node is a pixel, the association weights will depend on the difference in colour and/or texture and/or intensity. The association weights between a pixel and its neighbours may be large, but the association weights will die off quickly with distance, because there needs to be more evidence than just similarity of colour to say that two widely separated pixels belong together. As a result, we can reasonably expect to be dealing with clusters that are (a) quite tight and (b) distinct.

These properties lead to a fairly characteristic structure in the affinity matrix. In particular, if we relabel the nodes of the graph, then the rows and columns of the matrix $\mathcal{A}$ are shuffled. We expect to be dealing with relatively few collections of nodes with large association weights; furthermore, that these collections actually form a series of relatively coherent, largely disjoint clusters. This means that we could shuffle the rows and columns of M to form a matrix that is roughly block-diagonal (the blocks being the clusters). Shuffling M simply shuffles the elements

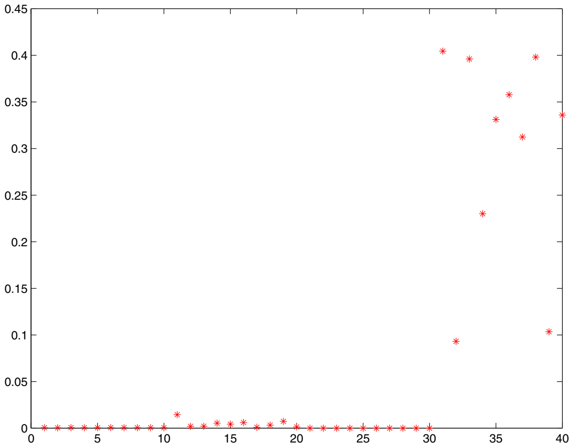


Figure 15.21. The eigenvector corresponding to the largest eigenvalue of the affinity matrix for the dataset of example 15.20, using $\sigma_d = 0.2$. Notice that most values are small, but some — corresponding to the elements of the main cluster — are large. The sign of the association is not significant, because a scaled eigenvector is still an eigenvector.

of its eigenvectors, so that we can reason about the eigenvectors by thinking about a shuffled version of M (i.e. figure 15.18 is a fair source of insight).

The eigenvectors of block-diagonal matrices consist of eigenvectors of the blocks, padded out with zeros. We expect that each block has an eigenvector corresponding to a rather large eigenvalue — corresponding to the cluster — and then a series of small eigenvalues of no particular significance. From this, we expect that, if there are c significant clusters (where $c < k$), the eigenvectors corresponding to the c largest eigenvalues each represent a cluster.

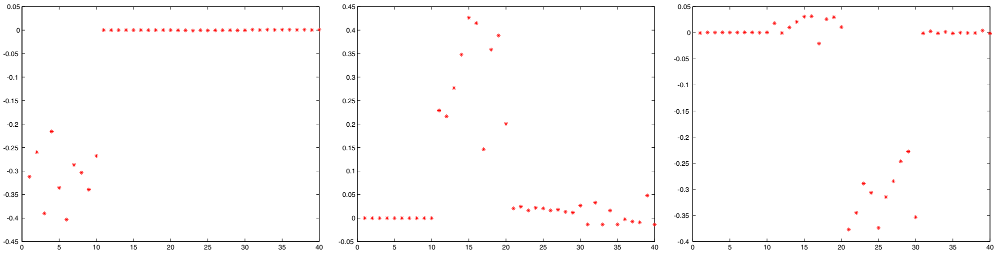


Figure 15.22. The three eigenvectors corresponding to the next three largest eigenvalues of the affinity matrix for the dataset of example 15.20, using $\sigma_d = 0.2$ (the eigenvector corresponding to the largest eigenvalue is given in figure 15.21). Notice that most values are small, but for (disjoint) sets of elements, the corresponding values are large. This follows from the block structure of the affinity matrix. The sign of the association is not significant, because a scaled eigenvector is still an eigenvector.

This means that each of these eigenvectors is an eigenvector of a block, padded with zeros. In particular, a typical eigenvector will have a small set of large values — corresponding to its block — and a set of near-zero values. We expect that only one of these eigenvectors will have a large value for any given component; all the others will be small (figure 15.22). Thus, we can interpret eigenvectors corresponding to the c largest magnitude eigenvalues as cluster weights for the first c clusters. One can usually quantize the cluster weights to zero or one, to obtain discrete clusters; this is what has happened in the figures.

```
Construct an affinity matrix

Compute the eigenvalues and eigenvectors of the affinity matrix

Until there are sufficient clusters

    Take the eigenvector corresponding to the
    largest unprocessed eigenvalue; zero all components corresponding
    to elements that have already been clustered, and threshold the
    remaining components to determine which element
    belongs to this cluster, choosing a threshold by
    clustering the components, or
    using a threshold fixed in advance.

    If all elements have been accounted for, there are
    sufficient clusters

end
```

Algorithm 15.6: *Clustering by Graph Eigenvectors*

This is a qualitative argument, and there are graphs for which the argument is decidedly suspect. Furthermore, we have been decidedly vague about how to determine c, though our argument suggests that poking around in the spectrum of $\mathcal{A}$ might be rewarding — one would hope to find a small set of large eigenvalues, and a large set of small eigenvalues (figure 15.23).

15.4.5 Normalised Cuts

The qualitative argument of the previous section is somewhat soft. For example, if the eigenvalues of the blocks are very similar, we could end up with eigenvectors that do not split clusters, because any linear combination of eigenvectors with the same eigenvalue is also an eigenvector (figure 15.24).

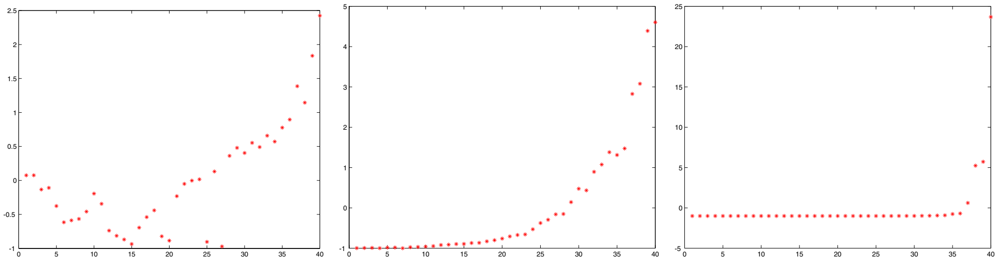


Figure 15.23. The number of clusters is reflected in the eigenvalues of the affinity matrix. The figure shows eigenvalues of the affinity matrices for each of the cases in figure 15.20. On the **left**, $\sigma_d = 0.1$, in the **center** $\sigma_d = 0.2$ and on the **right**, $\sigma_d = 1$. For the finest scale, there are many rather large eigenvalues — this is because the affinity between all points is rather small; for the next scale, there are four eigenvalues rather larger than the rest; and for the coarsest scale, there are only two eigenvalues rather larger than the rest.

An alternative approach is to cut the graph into two connected components such that the cost of the cut is a small fraction of the total affinity within each group. We can formalise this as decomposing a weighted graph V into two components A and B, and scoring the decomposition with

$$\frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

(where $cut(A,B)$ is the sum of weights of all edges in V that have one end in A and the other in B, and $assoc(A,V)$ is the sum of weights of all edges that have one end in A). This score will be small if the cut separates two components that have very few edges of low weight between them and many internal edges of high weight. We would like to find the cut with the minimum value of this criterion, called a **normalized cut**.

This problem is too difficult to solve in this form, because we would need to look at every graph cut — it's a combinatorial optimization problem, so we can't use continuity arguments to reason about how good a neighbouring cut is given the value of a particular cut. However, by introducing some terminology we can construct an approximation algorithm that generates a good cut.

We write $\boldsymbol{y}$ is a vector of elements, one for each graph node, *whose values are either* 1 *or* $-b$. The values of $\boldsymbol{y}$ are used to distinguish between the components of the graph: if the i'th component of $\boldsymbol{y}$ is 1, then the corresponding node in the graph belongs to one component, and if it is $-b$, the node belongs to the other. We write the affinity matrix as $\mathcal{A}$ is the matrix of weights between nodes in the graph and $\mathcal{D}$ is the **degree matrix**; each diagonal element of this matrix is the sum of weights coming into the corresponding node, that is

$$D_{ii} = \sum_j A_{ij}$$

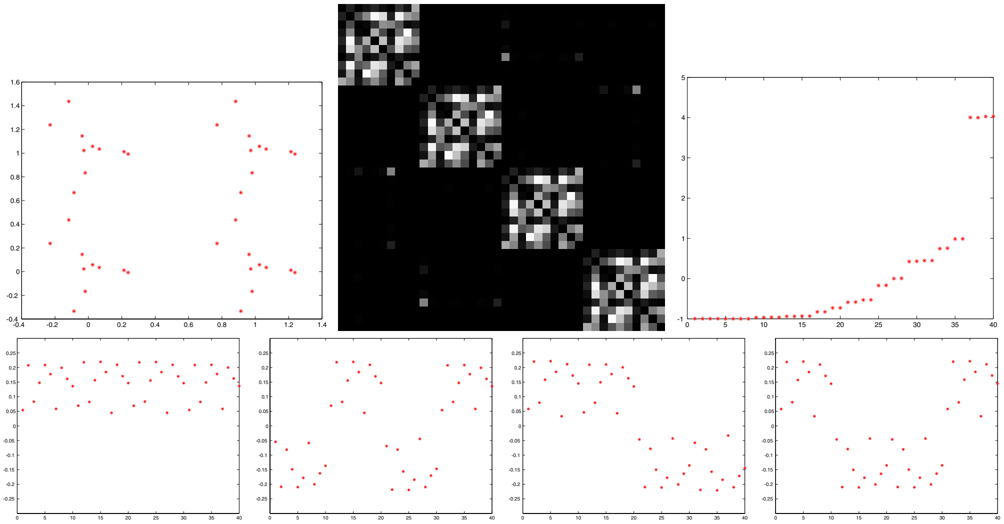


Figure 15.24. Eigenvectors of an affinity matrix can be a misleading guide to clusters. The dataset on the **top left** consists of four copies of the same set of points; this leads to a repeated block structure in the affinity matrix shown in the **top center**. Each block has the same spectrum, and this results in a spectrum for the affinity matrix that has (roughly) four copies of the same eigenvalue (**top right**). The bottom row shows the eigenvectors corresponding to the four largest eigenvalues; notice (a) that the values don't suggest clusters and (b) a linear combination of the eigenvectors might lead to a quite good clustering.

and the off-diagonal elements of $\mathcal{D}$ are zero. In this notation, and with a little manipulation, our criterion can be rewritten as:

$$\frac{\boldsymbol{y}^T(\mathcal{D}-\mathcal{A})\boldsymbol{y}}{\boldsymbol{y}^T\mathcal{D}\boldsymbol{y}}$$

We now wish to find a vector $\boldsymbol{y}$ that minimizes this criterion. The problem we have set up is an **integer programming** problem, and because it is exactly equivalent to the graph cut problem, it isn't any easier. The difficulty is the discrete values for elements of $\boldsymbol{y}$ — in principle, we could solve the problem by testing every possible $\boldsymbol{y}$, but this involves searching a space whose size is exponential in the number of pixels which will be slow[2]. A common approximate solution to such problems is to compute a *real* vector $\boldsymbol{y}$ that minimizes the criterion. Elements are then assigned to one side or the other by testing against a threshold. There are then two issues: firstly, we must obtain the real vector, and secondly, we must choose a threshold.

---

[2] As in, probably won't finish before the universe burns out.

Obtaining a Real Vector

The real vector is easily obtained. It is an exercise to show that a solution to

$$(\mathcal{D} - \mathcal{A})\boldsymbol{y} = \lambda \mathcal{D}\boldsymbol{y}$$

is a solution to our problem *with real values*. The only question is which generalised eigenvector to use? It turns out that the smallest eigenvalue is guaranteed to be zero, so the eigenvector corresponding to the second smallest eigenvalue is appropriate. The easiest way to determine this eigenvector is to perform the transformation $\boldsymbol{z} = \mathcal{D}^{1/2}\boldsymbol{y}$, and so get:

$$\mathcal{D}^{-1/2}(\mathcal{D} - \mathcal{A})\mathcal{D}^{-1/2}\boldsymbol{z} = \lambda \boldsymbol{z}$$

and $\boldsymbol{y}$ follows easily. Note that solutions to this problem are also solutions to

$$\mathcal{N}\boldsymbol{z} = \mathcal{D}^{-1/2}\mathcal{A}\mathcal{D}^{-1/2}\boldsymbol{z} = \mu \boldsymbol{z}$$

and $\mathcal{N}$ is sometimes called the **normalised affinity matrix**.

Choosing a Threshold

Finding the appropriate threshold value is not particularly difficult; assume there are N nodes in the graph, so that there are N elements in $\boldsymbol{y}$, and at most N different values. Now if we write $ncut(v)$ for the value of the normalised cut criterion at a particular threshold value v, there are at most $N + 1$ values of $ncut(v)$. We can form each of these values, and choose a threshold that leads to the smallest. Notice also that this formalism lends itself to recursion, in that each component of the result is a graph, and these new graphs can be split, too. A simpler criterion, which appears to work in practice, is to walk down the eigenvalues and use eigenvectors corresponding to smaller eigenvalues to obtain new clusters.

15.5 Discussion

Segmentation is a difficult topic, and there are a huge variety of methods. Methods tend to be rather arbitrary — remember, this doesn't mean they're not useful — because there really isn't much theory available to predict what should be clustered and how. It is clear that what we should be doing is forming clusters that are helpful to a particular application, but this criterion hasn't been formalised in any useful way. In this chapter, we have attempted to give the big picture while ignoring detail, because a detailed record of what has been done would be unenlightening.

Segmentation is also a key open problem in vision, which is why a detailed record of what has been done would be huge. Up until quite recently, it was usual to talk about recognition and segmentation as if they were distinct activities. This view is going out of fashion — as it should — because there isn't much point in creating a segmented representation that doesn't help with some application; furthermore, if we can be crisp about what should be recognised, that should make it possible to be crisp about what a segmented representation should look like.

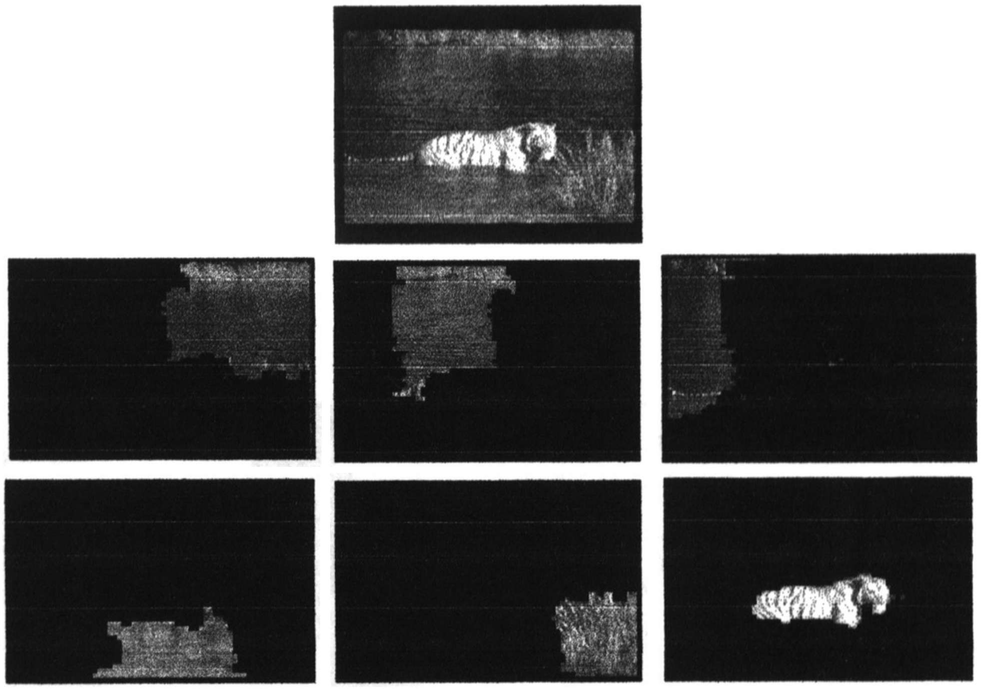


Figure 15.25. The image on top is segmented using the normalised cuts framework, described in the text, into the components shown. The affinity measures used involved intensity and texture, as in section 15.4.3. The image of the swimming tiger yields one segment that is essentially tiger, one that is grass, and four components corresponding to the lake. Note the improvement over k-means segmentation obtained by having a texture measure.

Assignments

Exercises

- We wish to cluster a set of pixels using colour and texture differences. The objective function

$$\Phi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{`th cluster}} (\boldsymbol{x}_j - \boldsymbol{c}_i)^T (\boldsymbol{x}_j - \boldsymbol{c}_i) \right\}$$

  used in section 15.3.3 may be inappropriate — for example, colour differences could be too strongly weighted if colour and texture are measured on different scales.

  1. Extend the description of the k-means algorithm to deal with the case

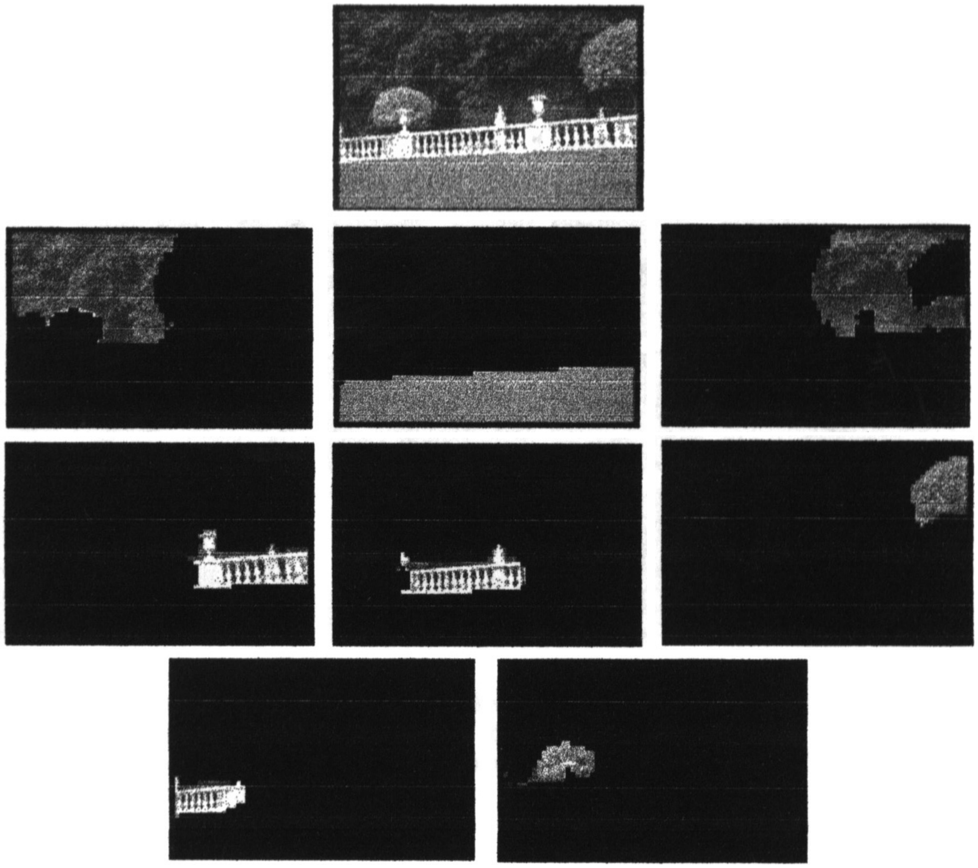


Figure 15.26. The image on top is segmented using the normalised cuts framework, described in the text, into the components shown. The affinity measures used involved intensity and texture, as in section 15.4.3. Again, note the improvement over k-means segmentation obtained by having a texture measure; the railing now shows as three reasonably coherent segments.

of an objective function of the form

$$\Phi(\text{clusters}, \text{data}) = \sum_{i \in \text{clusters}} \left\{ \sum_{j \in i\text{`th cluster}} (\boldsymbol{x}_j - \boldsymbol{c}_i)^T \mathcal{S} (\boldsymbol{x}_j - \boldsymbol{c}_i) \right\}$$

where $\mathcal{S}$ is an a symmetric, positive definite matrix.

2. For the simpler objective function, we had to ensure that each cluster contained at least one element (otherwise we can't compute the cluster center). How many elements must a cluster contain for the more

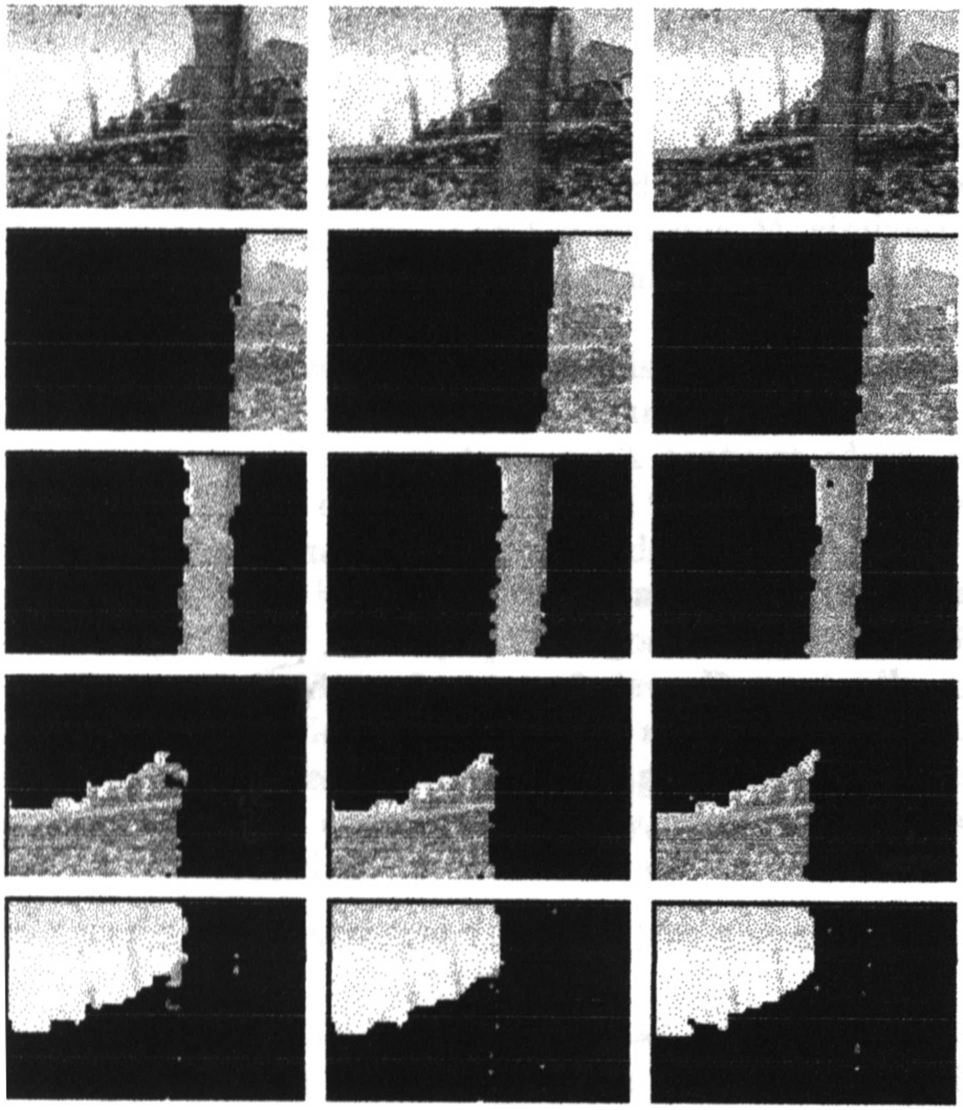


Figure 15.27. Three of the first six frames of a motion sequence, which shows a moving view of a house; the tree sweeps past the front of the house. Below, we see spatio-temporal segments established using normalised cuts and a spatio-temporal affinity function (section 15.4.3).

complicated objective function?

3. As we remarked in section 15.3.3, there is no guarantee that k-means

gets to a global minimum of the objective function; show that it must always get to a local minimum.

4. Sketch two possible local minima for a k-means clustering method clustering data points described by a two-dimensional feature vector. Use an example with only two clusters, for simplicity. You shouldn't need many data points. You should do this exercise for both objective functions.

- Read [**?**] and follow the proof that the normalised cut criterion leads to the integer programming problem given in the text. Why does the normalised affinity matrix have a null space? give a vector in its kernel.

- Show that choosing a *real* vector that maximises the expression

$$\frac{\boldsymbol{y}^T(\mathcal{D}-\mathcal{W})\boldsymbol{y}}{\boldsymbol{y}^T\mathcal{D}\boldsymbol{y}}$$

is the same as solving the eigenvalue problem

$$\mathcal{D}^{-1/2}\mathcal{W}\mathcal{W}\boldsymbol{z} = \mu\boldsymbol{z}$$

where $\boldsymbol{z} = \mathcal{D}^{-1/2}\boldsymbol{y}$.

- Grouping based on eigenvectors presents one difficulty: how to obtain eigenvectors for a large matrix quickly. The standard method is **Lanczos' algorithm**; read [], p.xxx-yyy, and implement this algorithm. Determine the time taken to obtain eigenvectors for a series of images of different sizes. Is your data consistent with the (known) order of growth of the algorithm?

- This exercise explores using normalised cuts to obtain more than two clusters. One strategy is to construct a new graph for each component separately, and call the algorithm recursively. You should notice a strong similarity between this approach and classical divisive clustering algorithms. The other strategy is to look at eigenvectors corresponding to smaller eigenvalues.

  1. Explain why these strategies are not equivalent.
  2. Now assume that we have a graph that has two connected components. Describe the eigenvector corresponding to the largest eigenvalue.
  3. Now describe the eigenvector corresponding to the second largest eigenvalue.
  4. Turn this information into an argument that the two strategies for generating more clusters should yield quite similar results under appropriate conditions; what are appropriate conditions?

- Show that the viewing cone for a cone is a family of planes, all of which pass through the focal point and the vertex of the cone. Now show the outline of a cone consists of a set of lines passing through a vertex. You should be able to do this by a simple argument, without any need for calculations.

Programming Assignments

- Build a background subtraction algorithm using a moving average and experiment with the filter.
- Build a shot boundary detection system using any two techniques that appeal, and compare performance on different runs of video.
- Implement a segmenter that uses k-means to form segments based on colour and position. Describe the effect of different choices of the number of segments; investigate the effects of different local minima.
- Implement a hough transform line finder.
- Count lines with an HT line finder - how well does it work?

Chapter 16

FITTING

There are a variety of segmentation criteria that involve models at a larger scale. Typically, one wants to decompose an image or a set of tokens — which could be pixels, isolated points, sets of edge points, etc. — into components that belong to one or another simple family. For example, we might want to cluster tokens together because they form a circle (which seems to be what's going on in figure 16.1). Finding such groups is often called **fitting**. We see fitting as part of the segmentation process, because it uses a model to produce compact representations that emphasize the relevant image structures.

Fitting lines as a model problem: there is a very wide range of possible fitting strategies; to keep track of what's important, and to see ideas in a reasonable context, we need a model problem. Fitting lines is a good model problem, because it is clear what the main issues are, and the technical aspects are relatively simple.

Generally, fitting involves determining what possible structures could have given rise to a set of tokens observed in an image. For example, we might have a set of edge points (the tokens) and wish to determine which lines fit them best. There are three quite general problems that occur in fitting:

- **Parameter estimation:** In this case, we assume we know *which tokens came from a particular structure*, and we want to know what the parameters of the structure are. For example, we might have a set of edge points, *all of which are known to have come from a line*, and we wish to know what line they came from.

- **Which token came from which structure:** In this case, we assume we know *how many structures are present*, and we wish to determine which tokens came from which structure. For example, we might have a set of edge points, and we need to know the best set of lines fitting these points; this involves (1) determining which points belong together on a line and (2) figuring out what each line is. Generally, these problems are not independent (because one good way of knowing whether points belong together on a line is checking how well the best fitting line approximates them).

- **Counting:** In this case, we would like to know (1) how many structures are present (2) which points are associated with which structure and (3) what the structures are. For example, given a set of edge points, we might want to return a set of lines that fits them well. This is, in general, a substantially difficult problem the answer to which depends strongly on the type of model adopted (for example, we could simply pass a line through every pair of edge points — this gives a set of lines that fit extremely well, but are a poor representation).

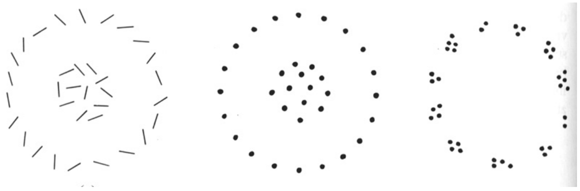


Figure 16.1. On occasion, tokens appear to be grouped together because they form useful primitives. For example, the main reason these tokens belong together appears to be that they form circles. *figure from Marr, Vision, page101, in the fervent hope that permission will be granted*

We first discuss a simple method for clustering tokens that lie on structures (section 16.1)— in practice, it is almost always used to find points that lie on lines — and then look at methods for fitting lines to point sets (section 16.2). We investigate fitting curves other than lines in section 16.3, and then discuss clustering edge points that could lie on the outlines of interesting objects (section 16.4).

16.1 The Hough Transform

One way to cluster points that could lie on the same structure is to record all the structures on which each point lies, and then look for structures that get many votes. This (quite general) technique is known as the **Hough transform**. We take each image token, and determine all structures *that could pass through that token.* We make a record of this set — you should think of this as voting — and repeat the process for each token. We decide on what is present by looking at the votes. For example, if we are grouping points that lie on lines, we take each point and vote for all lines that could go through it; we now do this for each point. The line (or lines) that are present should make themselves obvious, because they pass through many points and so have many votes.

16.1.1 Fitting Lines with the Hough Transform

Hough transforms tend to be most successfully applied to line finding. We will do this example to illustrate the method and its drawbacks. A line is easily parametrised as a collection of points (x, y) such that

$$x \cos\theta + y \sin\theta + r = 0$$

Now any pair of (θ, r) represents a unique line, where $r \geq 0$ is the perpendicular distance from the line to the origin, and $0 \leq \theta < 2\pi$. We call the set of pairs (θ, r) **line space**; the space can be visualised as a half-infinite cylinder. There is a family of lines that passes through any point token. In particular, the lines that lie on the curve *in line space* given by $r = -x_0 \cos\theta + y_0 \sin\theta$ all pass through the point token at (x_0, y_0).

Because the image has a known size, there is some R such that we are not interested in lines for $r > R$ — these lines will be too far away from the origin for us to see them. This means that the lines we are interested in form a bounded subset of the plane, and we discretize this with some convenient grid (which we'll discuss later). The grid elements can be thought of as buckets, into which we will sort votes. This grid of buckets is referred to as the **accumulator array**. Now for each point token we add a vote to the total formed for every grid element on the curve corresponding to the point token. If there are many point tokens that are collinear, we expect that there will be many votes in the grid element corresponding to that line.

16.1.2 Practical Problems with the Hough Transform

Unfortunately, the Hough transform comes with a number of important practical problems:

- **Quantization errors:** an appropriate grid size is difficult to pick. Too coarse a grid can lead to large values of the vote being obtained falsely, because many quite different lines correspond to a bucket. Too fine a value of the grid can lead to lines not being found, because votes resulting from tokens that are not exactly collinear end up in different buckets, and no bucket has a large vote (figure 16.2).

- **Difficulties with noise:** the attraction of the Hough transform is that it connects widely separated tokens that lie "close" to some form of parametric curve. This is also a weakness; it is usually possible to find many quite good phantom lines in a large set of reasonably uniformly distributed tokens. This means that, for example, regions of texture can generate peaks in the voting array that are larger than those associated with the lines sought (figures 16.4 and 16.5).

The Hough transform is worth talking about, because, despite these difficulties, it can often be implemented in a way that is quite useful for well-adapted problems. In practice, it is almost always used to find lines in sets of edge points. Useful implementation guidelines are:

- **Ensure the minimum of irrelevant tokens** this can often be done by tuning the edge detector to smooth out texture, setting the illumination to produce high contrast edges, etc.

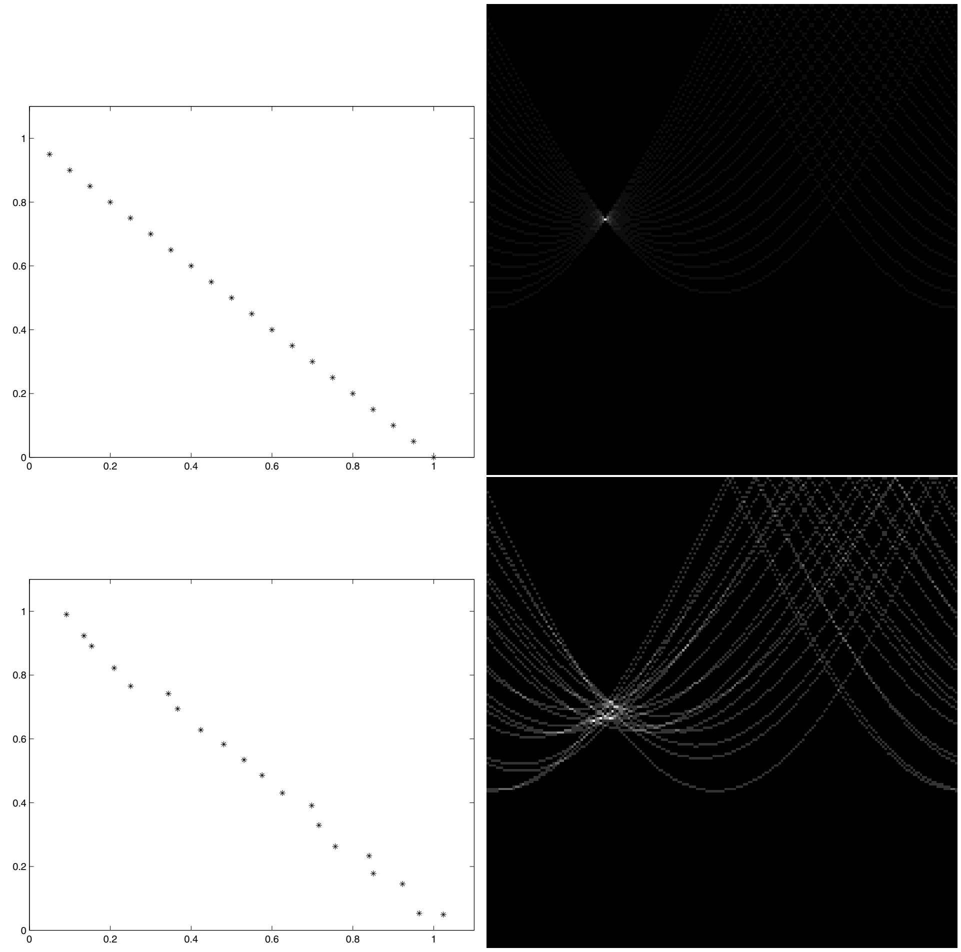


Figure 16.2. The Hough transform maps each point like token to a curve of possible lines (or other parametric curves) through that point. These figures illustrate the Hough transform for lines. The left hand column shows points, and the right hand column shows the corresponding accumulator arrays (the number of votes is indicated by the grey level, with a large number of votes being indicated by bright points). The top shows a set of 20 points drawn from a line next to the accumulator array for the Hough transform of these points. Corresponding to each point is a curve of votes in the accumulator array; the largest set of votes is 20. The horizontal variable in the accumulator array is θ and the vertical variable is r; there are 200 steps in each direction, and r lies in the range $[0, 1.55]$. In the center, these points have been offset by a random vector each element of which is uniform in the range $[0, 0.05]$; note that this offsets the curves in the accumulator array shown next to the points; the maximum vote is now 6.

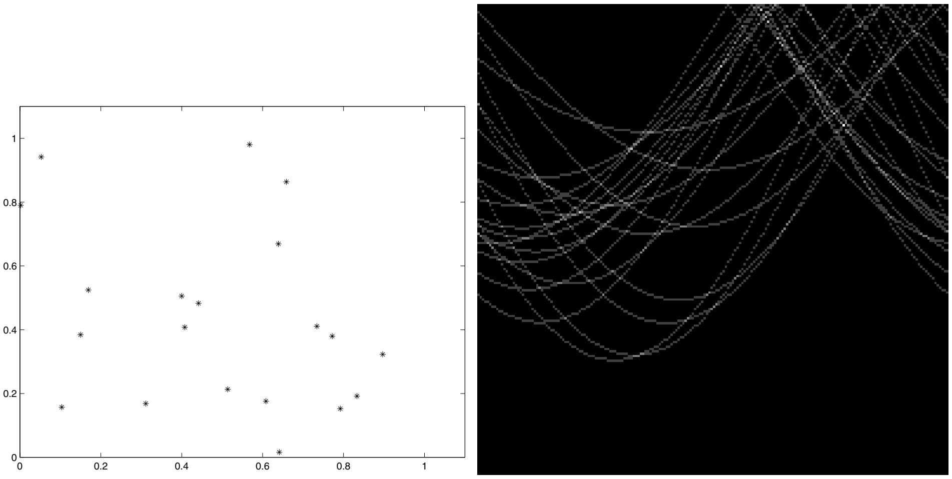


Figure 16.3. The Hough transform for a set of random points can lead to quite large sets of votes in the accumulator array. As in figure 16.2, the left hand column shows points, and the right hand column shows the corresponding accumulator arrays (the number of votes is indicated by the grey level, with a large number of votes being indicated by bright points). In this case, the data points are noise points (both coordinates are uniform random numbers in the range $[0, 1]$); the accumulator array in this case contains many points of overlap, and the maximum vote is now 4. Figures 16.4 and explore noise issues somewhat further.

- **Choose the grid carefully** this is usually done by trial and error. It can be helpful to vote for all neighbours of a grid element at the same time one votes for the element.

16.2 Fitting Lines

Line fitting is extremely useful. In many applications, objects are characterised by the presence of straight lines. For example, we might wish to build models of buildings using pictures of the buildings (as in the application in chapter **??**). This application uses polyhedral models of buildings, meaning that straight lines in the image are important. Similarly, many industrial parts have straight edges of one form or another, and if we wish to recognise industrial parts in an image, straight lines could be helpful. This suggests a segmentation that reports all straight lines in the image.

The first step in line fitting is to establish a probabilistic model that indicates how our data relates to any underlying line; with that model in place, we will proceed to determine how points can be allocated to particular lines, and how to count lines.

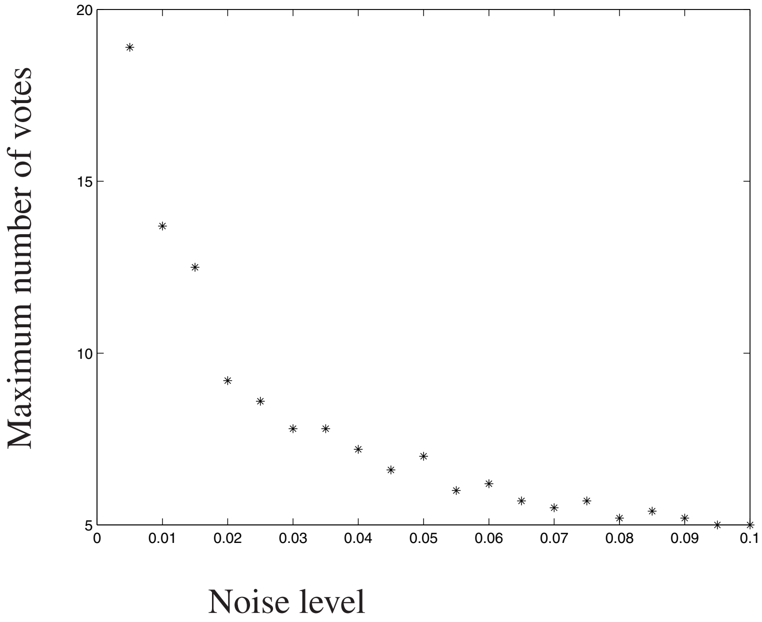


Figure 16.4. The effects of noise make it difficult to use a Hough transform robustly. The plot shows the maximum number of votes in the accumulator array for a Hough transform of 20 points on a line perturbed by uniform noise, plotted against the magnitude of the noise. The noise displaces the curves from each other, and quite quickly leads to a collapse in the number of votes. The plot has been averaged over 10 trials.

16.2.1 Least Squares, Maximum Likelihood and Parameter Estimation

Assume that all the points that belong to a particular line are known, and the parameters of the line must be found. We now need a **generative model** that indicates how our measurements were generated, given that the line was present. This generative model will give us an expression for the likelihood. In the vast majority of practical cases there is no reason to believe any one line is more likely to occur than any other, so that the distinction between maximum likelihood and MAP inference is moot. This means we wish to perform maximum likelihood inference. We adopt the notation that

$$\overline{u} = \frac{\sum u_i}{k}$$

to simplify the presentation.

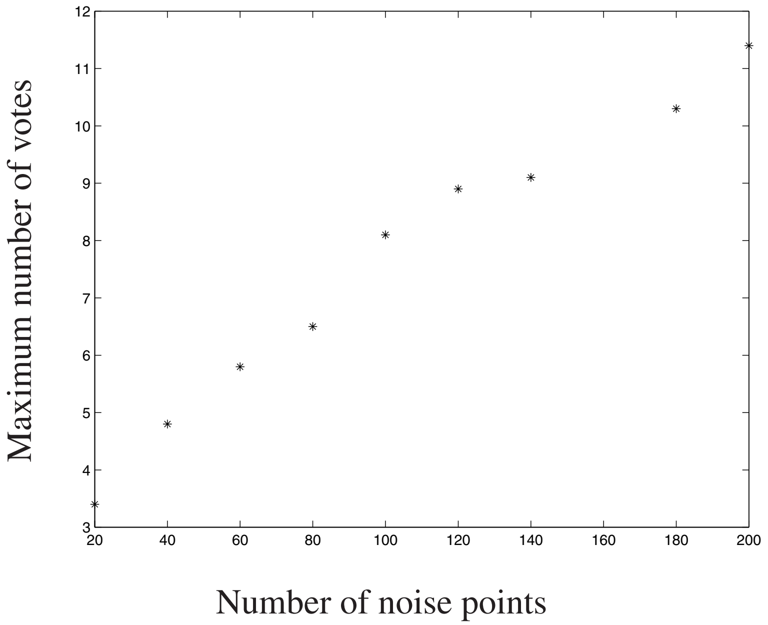


Figure 16.5. A plot of the maximum number of votes in the accumulator array for a Hough transform of a set of points whose coordinates are uniform random numbers in the range $[0, 1]$, plotted against the number of points. As the level of noise goes up, the number of votes in the right bucket goes down and the prospect of obtaining a large spurious vote in the accumulator array goes up. The plots have again been averaged over 10 trials. Compare this figure with figure 16.4, but notice the slightly different scales; the comparison suggests that it can be quite difficult to pull a line out of noise with a Hough transform (because the number of votes for the line might be comparable with the number of votes for a line due to noise). These figures illustrate the importance of ruling out as many noise tokens as possible before performing a Hough transform.

Total Least Squares

We can represent a line as the collection of points where $ax + by + c = 0$. Every line can be represented in this way, and we can think of a line as a triple of values (a, b, c). Notice that for $\lambda \neq 0$, the line given by $\lambda(a, b, c)$ is the same as the line represented by (a, b, c). Question **??** asks you to prove the simple, but extremely useful, result that the perpendicular distance from a point (u, v) to a line (a, b, c) is given by $\text{abs}(au + bv + c)$ *if* $a^2 + b^2 = 1$. In our experience, this fact is useful enough to be worth memorizing.

Generative model: We assume that our measurements are generated by choosing a point along the line, and then perturbing it perpendicular to the line using Gaussian noise. We assume that the process that chooses points along the line is uniform — in principle, it can't be, because the line is infinitely long, but in practice we can assume that any difference from uniformity is too small to bother with. This means we have a sequence of k measurements, (x_i, y_i), which are obtained from the model

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + n \begin{pmatrix} a \\ b \end{pmatrix}$$

where $n \sim N(0, \sigma)$, $au + bv + c = 0$ and $a^2 + b^2 = 1$.

Inference algorithm: The log-likelihood function is

$$-\frac{\sum_i (ax_i + by_i + c)^2}{2\sigma^2} + C$$

where $a^2 + b^2 = 1$ and C is some normalising constant of no interest. Thus, a maximum-likelihood solution is obtained by maximising this expression. Now using a Lagrange multiplier λ, we have a solution if

$$\begin{pmatrix} \overline{x^2} & \overline{xy} & \overline{x} \\ \overline{xy} & \overline{y^2} & \overline{y} \\ \overline{x} & \overline{y} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \lambda \begin{pmatrix} 2a \\ 2b \\ 0 \end{pmatrix}$$

This means that

$$c = -a\overline{x} - b\overline{y}$$

and we can substitute this back to get the eigenvalue problem

$$\begin{pmatrix} \overline{x^2} - \overline{x}\,\overline{x} & \overline{xy} - \overline{x}\,\overline{y} \\ \overline{xy} - \overline{x}\,\overline{y} & \overline{y^2} - \overline{y}\,\overline{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \mu \begin{pmatrix} a \\ b \end{pmatrix}$$

Because this is a 2D eigenvalue problem, two solutions up to scale can be obtained in closed form (for those who care - it's usually done numerically!). The scale is obtained from the constraint that $a^2 + b^2 = 1$. The two solutions to this problem are lines at right angles, and one maximises the likelihood and the other minimises it.

Least Squares

Least squares is a fitting procedure with a long tradition (which is the only reason we describe it!). It has the virtue of yielding a simple analysis and the very significant disadvantage of a generative model that very seldom makes sense in vision applications. For this approach, we represent a line as $y = ax + b$.

Generative model: The k measurements (x_i, y_i) are obtained from the model $y = ax + b + n$, where $n \sim N(0, \sigma)$. This means that only the y-coordinate of each measurement is affected by noise, which is why it is a rather dubious model.

Inference algorithm: The maximum likelihood estimate of a and b is easily obtained from the solution to

$$\begin{pmatrix} \overline{y^2} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} \overline{x^2} & \overline{x} \\ \overline{x} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

While this is a standard linear solution to a classical problem, it's actually not much help in vision applications because the model is an extremely poor model. The difficulty is that the measurement error is dependent on coordinate frame — we are counting vertical offsets from the line as errors, which means that near vertical lines lead to quite large values of the error and quite funny fits (figure 16.6). In fact, the process is so dependent on coordinate frame that it doesn't represent vertical lines at all.

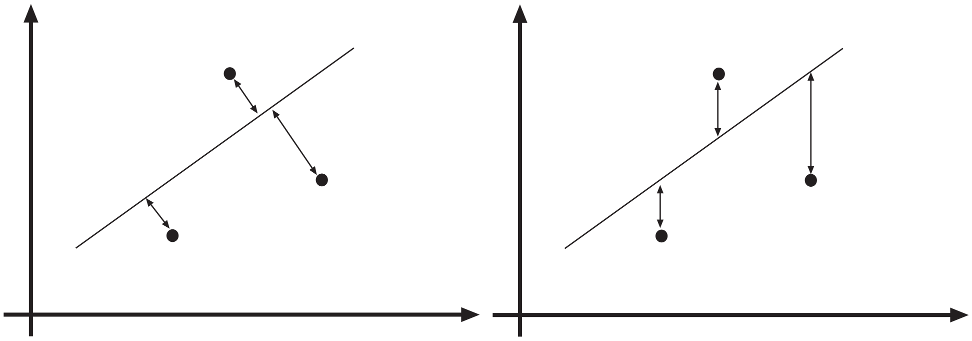


Figure 16.6. Left: Perpendicular least squares models data points as being generated by an abstract point along the line to which is added a vector perpendicular to the line, with a length given by a zero mean, Gaussian random variable. This means that the distance from data points to the line has a normal distribution. By setting this up as a maximum likelihood problem, we obtain a fitting criterion that chooses a line that minimizes the sum of distances between data points and the line. **Right:** Least squares follows the same general outline, but assumes that the error appears only in the y-coordinate. This yields a (very slightly) simpler mathematical problem, at the cost of a poor fit.

16.2.2 Which Point is on Which Line?

This problem can be very difficult, because it can involve search over a very large combinatorial space. One approach is to notice that we very seldom encounter isolated points; instead, we are fitting lines to edge points. We can use the orientation of an edge point as a hint to the position of the next point on the line. If we are stuck with isolated points, then both k-means and EM algorithms can be applied.

Incremental Fitting

Incremental line fitting algorithms take connected curves of edge points and fit lines to runs of points along the curve. Connected curves of edge points are fairly easily obtained from an edge detector whose output gives orientation (see exercises). An incremental fitter then starts at one end of a curve of edge points and walks along the curve, cutting off runs of pixels that fit a line well (the structure of the algorithm is shown in algorithm 1). Incremental line fitting can work very well indeed, despite the lack of an underlying statistical model. One feature is that it reports groups of lines that form closed curves. This is attractive when the lines one is interested in can reasonably be expected to form a closed curve (for example, in some object recognition applications)because it means that the algorithm reports natural groups without further fuss. This strategy often leads to occluded edges resulting in more than one fitted line. This difficulty can be addressed by postprocessing the lines to find pairs that (roughly) coincide, but the process is somewhat unattractive because it is hard to give a sensible criterion by which to decide when two lines do coincide.

```
Put all points on curve list, in order along the curve
empty the line point list
empty the line list

Until there are two few points on the curve
  Transfer first few points on the curve to the line point list
  fit line to line point list

  while fitted line is good enough
    transfer the next point on the curve
    to the line point list and refit the line
  end

  transfer last point back to curve
  attach line to line list
end
```

Algorithm 16.1: *Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large*

16.3 Fitting Curves

In principle, fitting curves is not very different from fitting lines. The usual generative model is that data points are generated uniformly at random on the curve, and then perturbed by Gaussian noise *normal* to the curve. This means that the distance between points and the underlying curve is normally distributed. In turn, a maximum likelihood approach minimizes the sum of distances between the points and the curve.

This generates quite difficult practical problems: it is usually very hard to tell the distance between a point and a curve. We can either solve this problem, or apply various approximations (which are usually chosen because they are computationally simple, not because they result from clean generative models). In this section, we will suppress the probabilistic issues — which were comprehensively discussed for the case of fitting lines — and concentrate on how to solve the distance problem for the two main representations of curves.

16.3.1 Implicit Curves

The coordinates of **implicit curves** satisfy some parametric equation; if this equation is a polynomial, then the curve is said to be **algebraic**, and this case is by far the most common. Some common cases are given in table 16.1.

Curve	equation
Line	$ax + by + c = 0$
Circle, center (a, b) and radius r	$x^2 + y^2 - 2ax - 2by + a^2 + b^2 - r^2 = 0$
Ellipses (including circles)	$ax^2 + bxy + cy^2 + dx + ey + f = 0$ where $b^2 - 4ac < 0$
Hyperbolae	$ax^2 + bxy + cy^2 + dx + ey + f = 0$ where $b^2 - 4ac > 0$
Parabolae	$ax^2 + bxy + cy^2 + dx + ey + f = 0$ where $b^2 - 4ac = 0$
General conic sections	$ax^2 + bxy + cy^2 + dx + ey + f = 0$

Table 16.1. Some implicit curves used in vision applications. Note that not all of these curves are guaranteed to have any real points on them — for example, $x^2 + y^2 + 1 = 0$ doesn't. Higher degree curves are seldom used, because it can be difficult to get stable fits to these curves.

The Distance from a Point to an Implicit Curve

Now we would like to know the distance from a data point to the closest point on the implicit curve. Assume that the curve has the form $\phi(x, y) = 0$. The vector from the closest point on the implicit curve to the data point is normal to the curve, so the closest point is given by finding all the (u, v) with the following properties:

1. (u, v) is a point on the curve — this means that $\phi(u, v) = 0$;
2. $\boldsymbol{s} = (d_x, d_y) - (u, v)$ is normal to the curve.

Given all such $\boldsymbol{s}$, the length of the shortest is the distance from the data point to the curve.

The second criterion requires a little work to determine the normal. The normal to an implicit curve is the direction in which we leave the curve fastest; along this direction, the value of ϕ must change fastest, too. This means that the normal at a point (u, v) is

$$(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y})$$

evaluated at (u, v). If the tangent to the curve is $\boldsymbol{T}$, then we must have $\boldsymbol{T}.\boldsymbol{s} = 0$. Because we are working in 2D, we can determine the tangent from the normal, so that we must have

$$\psi(u, v; d_x, d_y) = \frac{\partial \phi}{\partial y}(u, v)\,\{d_x - u\} - \frac{\partial \phi}{\partial x}(u, v)\,\{d_y - v\} = 0$$

at the point (u, v). We now have two equations in two unknowns, and *in principle* can solve them. However, this is very seldom as easy as it looks, as example 1 indicates.

A conic section is given by $ax^2 + bxy + cy^2 + dx + ey + f = 0$. Given a data point (d_x, d_y), the nearest point on the conic satisfies two equations:

$$\begin{aligned} au^2 + buv + cv^2 + du + ev + f &= 0 \\ &\text{and} \\ 2(a - c)uv - (2ad_y + e)u + (2cd_x + d)v + (ed_x - dd_y) &= 0 \end{aligned}$$

There can be up to four real solutions of this pair of equations (in the exercises, you are asked to demonstrate this, given an algorithm for obtaining the solutions, and asked to sketch various cases). As an example, choose the ellipse $2x^2 + y^2 - 1 = 0$, which yields the equations

$$2u^2 + v^2 - 1 = 0 \text{ and } 2uv - 4d_yu + 2d_xv = 0$$

Let us consider a family of data points $(d_x, d_y) = (0, \lambda)$; then we can rearrange these equations to get:

$$2u^2 + v^2 - 1 = 0 \text{ and } 2uv - 4\lambda u = 2u(v - 2\lambda) = 0$$

The second equation helps: either $u = 0$, or $v = 2\lambda$. Two of our solutions will be $(0, 1)$, $(0, -1)$. The other two are obtained by solving $2u^2 + 4\lambda^2 - 1 = 0$, which has solutions only if $-1/2 \leq \lambda \leq 1/2$. The situation is illustrated in figure 16.7.

Example 16.1: *The distance between a point and a conic*

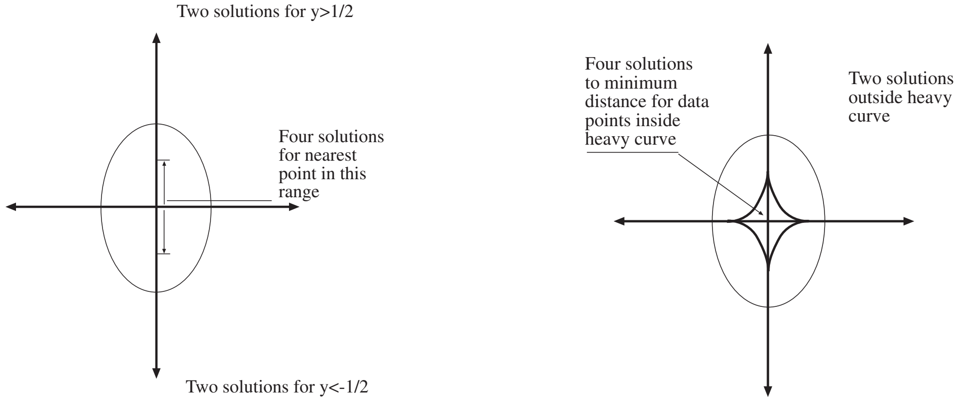

Two solutions for y>1/2
Four solutions for nearest point in this range
Two solutions for y<-1/2
Four solutions to minimum distance for data points inside heavy curve
Two solutions outside heavy curve

Figure 16.7. *On the left, the example worked in the text, where we study the number of possible solutions for the distance between a point and an ellipse for data points lying on the vertical axis. The figure on the right, indicates the general case for this ellipse.*

Approximations to the Distance

Notice that for a relatively simple curve we already have a somewhat nasty problem to solve. A curve with a slightly more complicated geometry — obtained by choosing ϕ to be a polynomial of higher degree, say d — leads to quite nasty problems. This is because the closest point on the curve would be obtained by solving two simultaneous polynomial equations, *both* of degree d. It can be shown that this can lead to as many as d^2 solutions, which are usually hard to obtain in practice. Various approximations to the distance between a point and an implicit algebraic curve have come into practice.

The best known is **algebraic distance**; in this case, we measure the distance between a curve and a point by evaluating the polynomial equation at that point, that is, we make the approximation:

$$\text{distance between } (d_x, d_y) \text{ and } \phi(x, y) = 0 = \phi(d_x, d_y)$$

This approximation can be (rather roughly!) justified when the data points are quite close to the curve. For a point sufficiently close to the curve *and to first order*, $\phi(d_x, d_y)$ increases as (d_x, d_y) moves normal to the curve — because the normal to the curve is given by the gradient of ϕ — and does not increase as (d_x, d_y) moves tangent to the curve. One significant difficulty is that, as it stands, algebraic distance is ill-defined, because many polynomials correspond to the same curve. In particular, the curve given by $\mu\phi(x, y) = 0$ is the same as the curve given by $\phi(x, y) = 0$. This problem can be solved normalising the coefficients of the polynomial in some way.

We have already seen one example of this process in section 16.2, where we fitted a line ($\phi(x, y) = ax + by + c = 0$) to a set of points by minimizing the algebraic distance, subject to the constraint that $a^2 + b^2 = 1$. In this case, the algebraic distance is the same as the actual distance. The choice of normalization is important. For example, if we try to fit conics ($ax^2+bxy+cy^2+dx+ey+f = 0$) using the constraint $b = 1$, we cannot fit circles.

An alternative approximation is to use

$$\frac{\phi(d_x, d_y)}{|\nabla\phi(d_x, d_y)|}$$

which has the advantage of not requiring a normalising constant; in the case of a line, this approximation is exact. Notice that this approximation has the same properties as algebraic distance — it goes up as one moves along the normal, etc. The advantage of the approximation is that is somewhat more accurate than algebraic distance, because it is normalised by the length of the normal. This means that it can be read — roughly! — as giving the percentage distance along the normal from the curve to the point. In practice, this approximation is seldom used, mainly because the use of algebraic distance yields simpler numerical problems.

Both of these approximations are very dangerous. This is because their behaviour for data points that are far from the curve is strange and not well understood. As a result, the relationship between a fitted curve and a set of data points becomes a bit mysterious if the data points don't lie very close to a curve of that class. Algebraic distance is used quite widely in practice, because it yields easy numerical problems and can be used for higher dimensional problems like approximating the distance between points and implicit surfaces. The exact distance is very difficult to compute for such problems.

16.3.2 Parametric Curves

The coordinates of a **parametric curve** are given as parametric functions of a parameter that varies along the curve. Parametric curves have the form:

$$(x(t), y(t)) = (x(t;\theta), y(t;\theta)) \qquad t \in [t_{min}, t_{max}]$$

Table 16.2 shows the form of a variety of useful parametric curves.

The Distance from a Point to a Parametric Curve

Assume we have a data point (d_x, d_y). The closest point on a parametric curve can be identified by its parameter value, which we shall write as τ. This point could lie at one or other end of the curve. Otherwise, the vector from our data point to the closest point is normal to the curve. This means that $\boldsymbol{s}(\tau) = (d_x, d_y) - (x(\tau), y(\tau))$ is normal to the tangent vector, so that $\boldsymbol{s}(\tau).\boldsymbol{T} = 0$. The tangent vector is

$$(\frac{dx}{dt}(\tau), \frac{dy}{dt}(\tau))$$

Curves	Parametric form	parameters
Circles centered at the origin	$(rsin(t), rcos(t))$	$\theta = r$ $t \in [0, 2\pi)$
Circles	$(rsin(t) + a, rcos(t) + b)$	$\theta = (r, a, b)$ $t \in [0, 2\pi)$
Axis aligned ellipses	$(r_1 sin(t) + a, r_2 cos(t) + b)$	$\theta = (r_1, r_2, a, b)$ $t \in [0, 2\pi)$
Ellipses	$(\cos\phi\,(r_1 sin(t) + a) - \sin\phi\,(r_2 cos(t) + b)\,,$ $\sin\phi\,(r_1 sin(t) + a) + \cos\phi\,(r_2 cos(t) + b))$	$\theta = (r_1, r_2, a, b, \phi)$ $t \in [0, 2\pi)$
cubic segments	$(at^3 + bt^2 + ct + d, et^3 + ft^2 + gt + h)$	$\theta = (a, b, c, d, e, f, g, h)$ $t \in [0, 1]$

Table 16.2. A selection of parametric curves often used in vision applications. It is quite common to put together a set of cubic curves, with constraints on their coefficients such that they form a single continuous differentiable curve; the result is known as a **cubic spline**.

which means that τ must satisfy the equation

$$\frac{dx}{dt}(\tau)\left\{d_x - x(\tau)\right\} + \frac{dy}{dt}(\tau)\left\{d_y - y(\tau)\right\} = 0$$

Now this is only one equation, rather than two, but the situation is not much better than that for parametric curves. It is almost always the case that $x(t)$ and $y(t)$ are polynomials, because it is usually easier to do root finding for polynomials. At worst, $x(t)$ and $y(t)$ are ratio's of polynomials, because we can rearrange the left hand side of our equation to come up with a polynomial in this case, too. However, we are still faced with a possibly large number of roots.

There is a second difficulty that makes fitting to parametric curves unpopular. Parametric curves with different coefficients may represent the same curve — for example, the curve $(x(t), y(t))$ for $t \in [0, 1]$ is the same as the curve $(x(2t), y(2t))$ for $t \in [0, 1/2]$. This situation can be very bad, depending on the class of parametric curves that we use (exercises).

16.4 Fitting to the Outlines of Surfaces

Generally, if we view a surface drawn from a constrained class — say, a surface of revolution — then its outline will satisfy some set of constraints, too. This observation is the key to a powerful idea: cluster edge points to form collections that "look like" the outline of a surface (i.e. satisfy the relevant constraints). The approach appears to work in practice only for quite simple cases, but some of these cases are important.

16.4.1 Some Relations Between Surfaces and Outlines

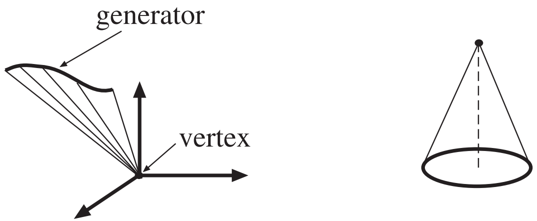
Recall that the outline of a surface is formed by slicing a cone of rays with the image plane. The cone of rays consists of rays tangent to the surface that pass through the focal point of the camera — for a perspective camera — or are parallel — for an affine camera. Call this cone the **viewing cone**. If the affine camera is orthographic, which is by far the most common case, then the slice is taken perpendicular to the rays. The viewing cone is usually easier to analyze than the outline.

Cones

A **cone** is a surface obtained by sweeping a family of rays through a point — the **vertex** of the cone — along a plane curve, called the **generator**. Notice that this definition is more general than that of a **right circular cone**, which many people incorrectly call a cone; right circular cones have a rotational symmetry (figure 16.8). A cone consists of scaled and translated copies of its generator. Choose a coordinate frame where the generator can be written as $(x(t), y(t), 1)$. Then the cone can be written as

$$(x(t)s, y(t)s, s)$$

and the vertex occurs at $(0, 0, 0)$ (figure 16.8).



A right circular cone

Figure 16.8. Cones are surfaces obtained by sweeping rays through a vertex along a generator. A right circular cone is a special cone, where the generator is a circle and the line joining the vertex with the center of the circle is normal to the circle's plane.

The viewing cone for a cone is a family of planes, all of which pass through the focal point and the vertex of the cone. This means that the outline of a cone consists of a set of lines passing through a vertex (figure 16.9). All this should be obvious (you are asked for a proof in the exercises), but is surprisingly useful.

Straight Homogenous Generalised Cylinders

A **straight homogenous generalised cylinder** (or **SHGC** for short) is a surface obtained by sweeping a plane generating curve along a line at right angles to the

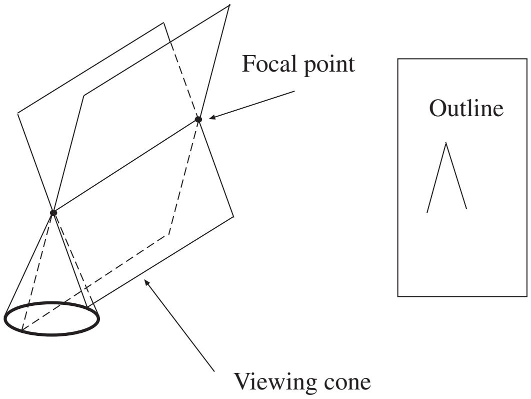


Figure 16.9. The viewing cone is a set of planes tangent to the cone, and passing through both the vertex of the cone and the focal point. The outline of a cone is obtained by slicing the viewing cone with a plane, and is a set of lines through a single point.

curve, and scaling it as one sweeps. With an appropriate choice of coordinate system, the generating curve is $(x(t), y(t), 0)$, and the surface can be written as

$$(x(t)f(s), y(t)f(s), g(s))$$

An SHGC is *locally* a cone. Fix a value of $s = s_0$ and consider the strip of surface from s_0 to $s_0 + \epsilon$. If the strip is small enough, then $f(s)$ can be approximated by $us + v$, and $g(s)$ can be approximated by $cs + d$, for u, v, c and d some constants (this is what derivatives are all about!). The d can be disposed of by translation and the c by reparametrisation, so this strip is a cone. This means that all the tangent vectors in the s direction at $s = s_0$ pass through some vertex.

It is a remarkable fact that the collection of vertices obtained for different values of s is collinear. The easiest way to see this is to work in coordinates. Take the coordinate form given above, and slice it with an arbitrary vertical plane, yielding two plane curves $(\sqrt{x(t_0)^2 + y(t_0)^2} f(s), g(s))$ and $(-\sqrt{x(t_1)^2 + y(t_1)^2} f(s), g(s))$. Take the tangents to these curves at $s = s_0$, and produce them to form tangent rays; a quick calculation shows that the rays meet at $(0, g(s_0) - (f(s_0)/f'(s_0))g'(s_0))$. This means that for any cross-section and any value of s, the tangents meet on the line $x = 0$, meaning that the vertices are collinear. Although we obtained this result in coordinates, this was merely for convenience. Because the result is about incidence properties of tangents — which aren't affected by change of coordinates — it applies to any SHGC in any coordinate frame. So for any SHGC, in any coordinate frame, there is a well defined axis, which is the line along which the vertices of the tangent cones fall.

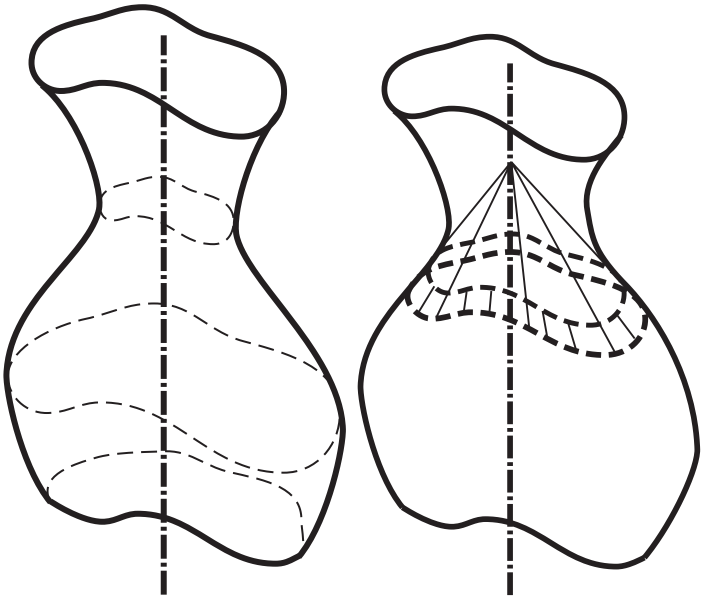


Figure 16.10. An SHGC is obtained by sweeping a plane generator along a line perpendicular to its plane, and growing or shrinking it as it sweeps (left). This gives a surface that is *locally* a cone, in the sense that there is a cone tangent to the surface along any generator. In the right hand figure, we have cut a narrow strip of the surface between two generators, and produced its tangents. These tangents meet at a vertex on the axis, meaning that each such strip of an SHGC is a strip of cone.

Surfaces of Revolution

A **surface of revolution** — or **SOR** for short — is a special SHGC, whose generator is a circle. The viewing cone for an SOR has a symmetry. Imagine the plane through the axis of the SOR and the focal point; the cone must have a flip symmetry about this plane. This *does not* mean that the image curve has this symmetry, because we are slicing the viewing cone with the image plane to get the image curve, and the slicing process can disrupt the symmetry (figure 16.12). The effect is governed by the field of view of the camera, and for the vast majority of practical cameras, it is tiny.

16.4.2 Clustering to Form Symmetries

Clustering the outline of a cone is relatively simple — a collection of image line segments that would pass through a single point, if extended, could be the outline of a cone.

Similarly, we can reason about the outline of an SHGC using our knowledge of cones. In particular, the tangents at components of the outline corresponding to the

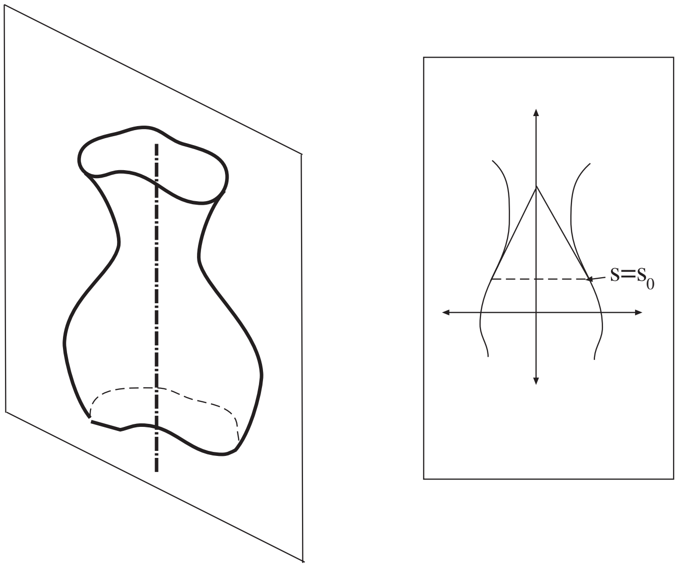


Figure 16.11. The vertices of the tangent cones to an SHGC all fall along a single line; this defines the axis for the SHGC. The easiest way to prove this is to work with the SHGC in the coordinatised form $(x(t)f(s), y(t)f(s), g(s))$, and slice it with an arbitrary plane through the z-axis (left). For any such slice, lines tangent to the cross-section at a particular s-value meet at a point on the z-axis, and the result follows.

same value of s, when produced, will meet at the projected vertex of the tangent cone for that value of s. In turn, there is some correspondence between components of the outline such that tangents at corresponding sides, when produced, will meet along a straight line. This is a segmentation criterion, because not all sets of curves will satisfy it. It is somewhat tricky to use in this form, however (but see [**?**; **?**]).

Surfaces of Revolution

Corresponding points on either side of the outline of an SOR can be identified by a local test on pairs of points on image curves. In particular, two points on image curves where the tangent is at about the same angle to the line joining the points (figure 16.13) could be on opposite sides of a symmetry — we will call this configuration a **local symmetry**, and the line segment joining the points the **symmetry line**.

We could find the outline of a surface of revolution by looking for local symmetries whose midpoints lie on a straight line roughly perpendicular to their symmetry lines. The main difficulty with this strategy is that most images contain an awful lot of symmetries, and there may be many groups of symmetries that satisfy this test.

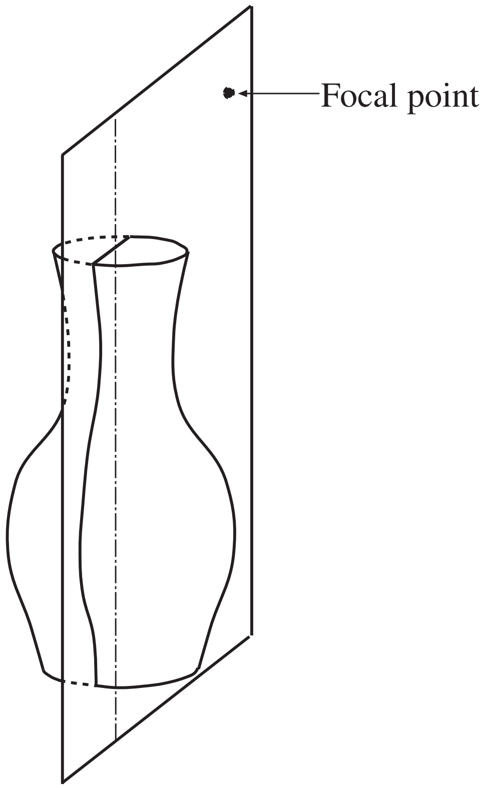


Figure 16.12. A surface of revolution and a focal point together give a plane of symmetry, that passes through the axis of the SOR and the focal point. The contour generator must have a mirror symmetry in this plane. This doesn't mean that the outline has an exact mirror symmetry, because the outline is obtained by slicing the viewing cone — which isn't shown, for simplicity — by a plane *that may not be at right angles to the plane of symmetry*. However, the effect is small, and to all intents and purposes the outline of an SOR can be regarded as having a mirror symmetry.

Cylinders and Body Segments

A particularly interesting SOR is a cylinder. This is because, to a crude first approximation human body segments look like this, as do body segments from some animals. In this case, the local symmetries will have midpoints that (roughly) lie on a straight line, *and* this line will be (roughly) perpendicular to the symmetry lines, *and* the lengths of all the symmetry lines will be (roughly) the same. While typical images contain an awful lot of local symmetries with these properties, it is just about practical to winnow through them looking for groups that satisfy these

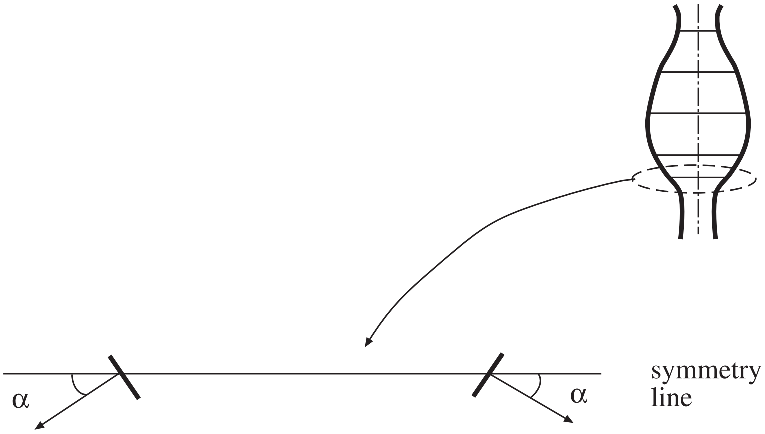


Figure 16.13. A local symmetry is a pair of contour points where the tangents to the contours are at roughly the same angle to the line joining the contours (the symmetry line). Such symmetries are often seen in the outlines of surfaces of revolution.

constraints.

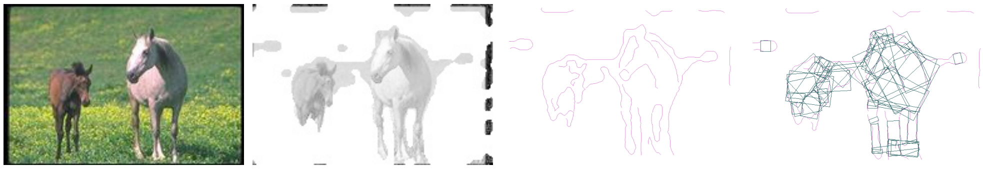


Figure 16.14. An example of the kind of representation that can be obtained from straight segments with near parallel sides. On the left, a colour image of two horses; on the center left, pixels with the right colour and texture to be hide have been retained and the others masked off; on the center right, the edges of this set of pixels; and on the far right, all straight segments with near parallel sides obtained using the mechanisms described above. These segments have been displayed using the abstraction maintained by the program (i.e. the sides of the abstract segment are shown superimposed on the edges that yield the segments). Notice that the components do not exactly correspond to body segments. Although it looks unpromising, this representation can be used to find the horses — which are represented as assemblies of segments — despite the fact it captures image information very poorly.

Cylinders can be found with a relatively crude algorithm (too crude to display!). We use a form of agglomerative clustering. Make each symmetry a cluster. We will build bigger clusters by looking forward and backward along the axis predicted by the symmetries in the cluster. Given a cluster, we can predict the orientation of the next symmetries in the cluster (roughly parallel to the symmetries in the cluster), and the position of their midpoints (along a line roughly perpendicular to the symmetry lines in the cluster), and their width (roughly the same as the width of the symmetries in the cluster). If the next symmetries are sufficiently nearby and sufficiently similar, we add them to the cluster, and proceed until the cluster cannot be grown further. We do this for each cluster. It is usually a good idea to have a second pass that engages in greedy merges between clusters, using the same criteria.

16.5 Discussion

Fitting should be seen as a form of segmentation exercise — we are concentrating attention on a set of image tokens that have some global structural properties. Section **??** gives the flavour of this line of reasoning, which is currently very poorly developed. We were deliberately vague in describing algorithms in that section; typically, words like "roughly" and "approximately" in our description are interpreted as tests against some threshold, which is set by hand. This means it is rather hard to be precise about what the representation means, or how well it performs. At the same time, symmetries do seem to yield representations that can be useful in practice — for example, it is possible to decide, with quite limited accuracy, whether a picture has unclad people in it or not using a representation based on symmetries **??**. The idea is useful because many interesting objects really do look like cylinders in images. Furthermore, cylinders are associated with a line of reasoning that is very attractive: we first check relatively small sets of image components to tell whether they are part of a cylinder, then assemble the survivors into a larger region. This means that we are using a model of what (some!) objects look like to help assemble together the evidence we need to tell whether an object is present. You should notice that, despite the fact that we have no real probabilistic model here, we seem to be doing something rather like inference: in the following chapter, we will describe the use of inference algorithms in segmentation.

Assignments

Exercises

- Prove the simple, but extremely useful, result that the perpendicular distance from a point (u, v) to a line (a, b, c) is given by $\text{abs}(au + bv + c)$ *if* $a^2 + b^2 = 1$.

- Derive the eigenvalue problem

$$\begin{pmatrix} \overline{x^2} - \overline{x}\,\overline{x} & \overline{xy} - \overline{x}\,\overline{y} \\ \overline{xy} - \overline{x}\,\overline{y} & \overline{y^2} - \overline{y}\,\overline{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \mu \begin{pmatrix} a \\ b \end{pmatrix}$$

  from the generative model for total least squares. This is a simple exercise — maximum likelihood and a little manipulation will do it — but worth doing right and remembering; the technique is extremely useful.

- How do we get a curve of edge points from an edge detector that returns orientation? - give a recursive algorithm.
- A slightly more stable variation of incremental fitting cuts the first few pixels and the last few pixels from the line point list when fitting the line, because these pixels may have come from a corner
  1. Why would this lead to an improvement?
  2. How should one decide how many pixels to omit?
- A conic section is given by $ax^2 + bxy + cy^2 + dx + ey + f = 0$.
  1. Given a data point (d_x, d_y), show that the nearest point on the conic (u, v) satisfies two equations:

     $$\begin{aligned} au^2 + buv + cv^2 + du + ev + f &= 0 \\ &\text{and} \\ 2(a-c)uv - (2ad_y + e)u + (2cd_x + d)v + (ed_x - dd_y) &= 0 \end{aligned}$$

  2. These are two quadratic equations. Write $\boldsymbol{u}$ for the vector $(u, v, 1)$. Now show that we can write these equations as $\boldsymbol{u}^T \mathcal{M}_1 \boldsymbol{u} = 0$ and $\boldsymbol{u}^T \mathcal{M}_2 \boldsymbol{u} = 0$, for $\mathcal{M}_1$ and $\mathcal{M}_2$ symmetric matrices.
  3. Show that there is a transformation $\mathcal{T}$, such that $\mathcal{T}^T \mathcal{M}_1 \mathcal{T} = Id$ *and* $\mathcal{T}^T \mathcal{M}_2 \mathcal{T}$ is diagonal.
  4. Now show how to use this transformation to obtain a set of solutions to the equations; in particular, show that there can be up to four real solutions.
  5. Show that there are either four, two or zero real solutions to these equations.
  6. Sketch an ellipse, and indicate the points for which there are four or two solutions.
- Show that the curve

  $$(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2})$$

  is a circular arc (the length of the arc depending on the interval for which the parameter is defined).
  1. Write out the equation in t for the closest point on this arc to some data point (d_x, d_y); what is the degree of this equation? How many solutions in t could there be?
  2. Now substitute $s^3 = t$ in the parametric equation, and write out the equation for the closest point on this arc to the same data point. What is the degree of the equation? why is it so high? What conclusions can you draw?

- Show that the outline of a cone consists of a set of lines passing through the projection of the vertex (you can do this in a short sentence if you think about it; it's a bad idea to try and do it with equations).

Programming Assignments

- Implement an incremental line fitter. Determine how significant a difference results if you leave out the first few pixels and the last few pixels from the line point list (put some care into building this, as it's a useful piece of software to have lying around in our experience).

Chapter 17

SEGMENTATION AND FITTING USING PROBABILISTIC METHODS

All the segmentation algorithms we described in the previous chapter involve essentially local models of similarity. Even though some algorithms attempt to build clusters that are good globally, the underlying *model* of similarity compares individual pixels. Furthermore, none of these algorithms involved an explicit probabilistic model of how measurements differed from the underlying abstraction that we are seeking.

We shall now look at explicitly probabilistic methods for segmentation. These methods attempt to explain data using models that are global. These models will attempt to explain a large collection of data with a small number of parameters. For example, we might take a set of tokens and fit a line to them; or take a pair of images and attempt to fit a parametric set of motion vectors that explain how pixels move from one to the other.

The first important concept is that of a missing data problem. In many segmentation problems, there are several possible sources of data (for example, a token might come from a line, or from noise); if we knew from which source the data had come (i.e. whether it came from the line, or from noise), the segmentation problem would be easy. In section 17.1, we deal with a number of segmentation problems by phrasing them in this form, and then using a general algorithm for missing data problems.

We then set up the problem of fitting lines to a set of tokens as a quite general example of a probabilistic fitting problem (section 16.2), and discuss methods for attacking this problem. This leads us to situations where occasional data items are hugely misleading, and we discuss methods for making fitting algorithms robust (section 17.2). We sketch how to generalize our line fitting techniques to handle curve fitting problems in section 16.3. Finally, we discuss methods for determining how many elements (lines, curves, segments, etc.) to fit to a particular data set (section 17.3).

17.1 Missing Data Problems, Fitting and Segmentation

Segmentation can rather naturally be phrased as a missing data problem. In this approach, there are a series of different possible sources of pixel values, and the missing data is the source from which the pixel was drawn. A number of other interesting problems can be phrased in this way, too, and there is a standard, quite simple, algorithm for this problem.

17.1.1 Missing Data Problems

It is quite common to encounter problems which contain missing data. There are two contexts: in the first, some terms in a data vector are missing for some instances and present for other (perhaps someone responding to a survey was embarrassed by a question); in the second, which is far more common in our applications, an inference problem can be made very much simpler by rewriting it using some variables whose values are unknown. We will demonstrate this method and appropriate algorithms with two examples.

Example: Owls and Rats

In a study area, there are g different species of rat. A rat of the l'th species appears in an owl's diet with probability π_l It is hard to observe owls eating rats. Instead, owl pellets — the indigestible bits of rat, regurgitated by owls — are found and rat skulls in the pellets are measured. These measurements form observations of a random vector $\boldsymbol{W}$. The j'th observation is $\boldsymbol{W}_j$. The conditional probability of observing measurements of a rat skull in a pellet given it comes from species l is known to be $f_l(\boldsymbol{W})$.

All this means that the probability density for a set of measurements of a rat skull is:

$$p(\boldsymbol{W}) = \sum_l \pi_l f_l(\boldsymbol{W})$$

This is a form of probability model that we shall encounter quite often. It is often referred to as a **mixture model**, because it consists of a finite mixture of distributions (the coefficients of the distributions are often referred to as **mixing weights** — clearly, they must sum to one). A sample can be drawn from this model by selecting the l'th component with probability π_l, and then drawing a sample from $f_l(\boldsymbol{W})$.

Under this model, the likelihood of a set of observations is:

$$\prod_{j \in \text{observations}} \left(\sum_{l=1}^{g} \pi_l f_l(\boldsymbol{W}_j) \right)$$

We would like to infer π_l. If we knew which species of rat was represented by each skull, the problem would be easy; we could estimate π_l by counting the number of rat skulls of the l'th species, and dividing by the total number of skulls. The difficulty is that we have only the measurements of the skulls, not the species of the rat associated with the skull.

Example: Image Segmentation

(This is a formulation due to Malik and his students []) At each pixel in an image, we compute a d-dimensional feature vector $\boldsymbol{x}$, which encapsulates position, colour and texture information. This feature vector could contain various colour representations, and the output of a series of filters centered at a particular pixel. Our image model is that each pixel is produced by a density associated by an image segment. This means that the image is produced by a mixture model, which consists of a weighted sum of g Gaussian densities. This model is a density in feature vector space that consists of a set of "blobs", each of which is associated with an image segment. We should like to determine: (1) the parameters of each of these blobs; (2) the mixing weights and (3) from which component each pixel came (thereby segmenting the image).

We encapsulate these parameters into a parameter vector, writing the mixing weights as α_l and the parameters of each blob as $\theta_l = (\boldsymbol{\mu}_l, \Sigma_l)$, to get $\Theta = (\alpha_1, \ldots, \alpha_g, \theta_1, \ldots, \theta_g)$. The mixture model then has the form

$$p(\boldsymbol{x}|\Theta) = \sum_{l=1}^{g} \alpha_l p_l(\boldsymbol{x}|\theta_l)$$

Again, this model is sampled by choosing a component and then sampling that component.

Each component density is the usual Gaussian:

$$p_l(\boldsymbol{x}|\theta_l) = \frac{1}{(2\pi)^{d/2} \det(\Sigma_i)^{1/2}} \exp\left\{ -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_i) \right\}$$

The likelihood function for an image is:

$$\prod_{j \in \text{observations}} \left(\sum_{l=1}^{g} \alpha_l p_l(\boldsymbol{x}_j|\theta_l) \right)$$

Each component is associated with a segment, and Θ is unknown.

If we knew the component from which each pixel came, then it would be simple to determine Θ. We could use maximum likelihood estimates for each θ_l, and then the fraction of the image in each component would give the α_l. Similarly, if we knew Θ, then for each pixel, we could determine the component that is most likely to have produced that pixel — this yields an image segmentation. The difficulty is that we know neither.

Strategy

For each of these examples, if we know the missing data then we can estimate the parameters effectively. Similarly, if we know the parameters, the missing data will follow. This suggests an iterative algorithm:

1. Obtain some estimate of the missing data, using a guess at the parameters;
2. now form a maximum likelihood estimate of the free parameters using the estimate of the missing data.

and we would iterate this procedure until (hopefully!) it converged. In the case of the owls and the rats, this would look like:

1. Obtain some estimate of the number of rats of each species in each owl pellet, using a guess at π_l;
2. now form a revised estimate of π_l using the number of rats of each species in each pellet.

For image segmentation, this would look like:

1. Obtain some estimate of the component from which each pixel's feature vector came, using an estimate of the θ_l.
2. Now update the θ_l, using this estimate.

17.1.2 The EM Algorithm

Although it would be nice if the procedures given above converged, there is no particular reason to believe that they do. In fact, given appropriate choices in each stage, they do. This is most easily shown by showing that they are examples of a general algorithm, the **expectation-maximization** algorithm.

A Formal Statement of Missing Data Problems

Assume we have two spaces, the **complete data space** $\mathcal{X}$ and the **incomplete data space** $\mathcal{Y}$. There is a map f, which takes $\mathcal{X}$ to $\mathcal{Y}$. This map "loses" the missing data; for example, it could be a projection. For the example of the owls and rats, the complete data space consists of the measurements of the skulls *and* a set of variables indicating from which type of rat the skull came; the incomplete data is obtained by dropping this second set of variables. For the example of image segmentation, the complete data consists of the measurements at each pixel *and* a set of variables indicating from which component of the mixture the measurements came; the incomplete data is obtained by dropping this second set of variables.

There is a parameter space $\mathcal{U}$. For the owls and rats, the parameter space consists of the mixing weights; for the image segmentation example, the parameter space consists of the mixing weights and of the parameters of each mixture component. We wish to obtain a maximum likelihood estimate for these parameters, given only incomplete data. If we had complete data, we could use the probability density function for the complete data space, written $p_c(\boldsymbol{x};\boldsymbol{u})$. The complete data log-likelihood is

$$\begin{aligned} L_c(\boldsymbol{x};\boldsymbol{u}) &= \log\{\prod_j p_c(\boldsymbol{x}_j;\boldsymbol{u})\} \\ &= \sum_j \log\left(p_c(\boldsymbol{x}_j;\boldsymbol{u})\right) \end{aligned}$$

In either of our examples, this log-likelihood would be relatively easy to work with. In the case of the owls and rats, the problem would be to estimate the mixing weights, given the type of rat from which each skull came. In the case of image segmentation, the problem would be to estimate the parameters for each image segment, given the segment from which each pixel came.

The problem is that we don't have the complete data. The probability density function for the *incomplete* data space is $p_i(\boldsymbol{y};\boldsymbol{u})$. Now

$$p_i(\boldsymbol{y};\boldsymbol{u}) = \int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}} p_c(\boldsymbol{x};\boldsymbol{u})d\boldsymbol{x}$$

that is, the probability density function for the incomplete data space is obtained by integrating the probability density function for the complete data space over all values that give the same $\boldsymbol{y}$. The incomplete data likelihood is

$$\prod_{j\in\text{observations}} p_i(\boldsymbol{y}_j;\boldsymbol{u})$$

We could form a maximum likelihood estimate for $\boldsymbol{u}$, given $\boldsymbol{y}$ by writing out the likelihood and maximising it. This isn't easy, because both the integral and the maximisation can be quite difficult to do. The usual strategy of taking logs doesn't make things easier, because of the integral *inside* the log. We have:

$$\begin{aligned} L_i(\boldsymbol{y};\boldsymbol{u}) &= \log\left\{\prod_j p_i(\boldsymbol{y}_j;\boldsymbol{u})\right\} \\ &= \sum_j \log\left(p_i(\boldsymbol{y}_j;\boldsymbol{u})\right) \\ &= \sum_j \log\left(\int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}_j} p_c(\boldsymbol{x};\boldsymbol{u})d\boldsymbol{x}\right) \end{aligned}$$

This form of expression is difficult to deal with. The reason that we are stuck with the incomplete data likelihood is that we don't know which of the many possible $\boldsymbol{x}$'s that could correspond to the $\boldsymbol{y}$'s that we observe actually does correspond. Forming the incomplete data likelihood involves averaging over all such $\boldsymbol{x}$'s.

The key idea in E-M is to obtain a working value for this $\boldsymbol{x}$ by computing an expectation. In particular, we will fix the parameters at some value, and then compute the expected value of $\boldsymbol{x}$, given the value of $\boldsymbol{y}$ and the parameter values. We then plug the expected value of $\boldsymbol{x}$ into the complete data log-likelihood, which is much easier to work with, and obtain a value of the parameters by maximising that. Now at this point, the expected value of $\boldsymbol{x}$ may have changed. We obtain an algorithm by alternating the expectation step with the maximisation step, and iterate until convergence.

More formally, given $\boldsymbol{u}^s$, we form $\boldsymbol{u}^{s+1}$ by:

1. Computing an expected value for the *complete* data using the incomplete data and the current value of the parameters. This estimate is given by:

   $$\overline{\boldsymbol{x}}_j^s = \int_{\boldsymbol{x}|f(\boldsymbol{x})=\boldsymbol{y}_j} \boldsymbol{x} p_c(\boldsymbol{x}; \boldsymbol{u}^s) d\boldsymbol{x}$$

   This is referred to as **the E-step**.

2. Maximizing the *complete* data log likelihood with respect to $\boldsymbol{u}$, using the expected value of the complete data computed in the E-step. That is, we compute

   $$\boldsymbol{u}^{s+1} = \arg\max_{\boldsymbol{u}} L_c(\overline{\boldsymbol{x}}^s; \boldsymbol{u})$$

   This is known as **the M-step**.

It can be shown that the incomplete data log-likelihood is increased at each step, meaning that the sequence $\boldsymbol{u}^s$ converges to a (local) maximum of the incomplete data log-likelihood (e.g. []). Of course, there is no guarantee that this algorithm converges to the *right* local maximum, and in some of the examples below we will show that finding the right local maximum can be a nuisance.

Example: Owls and Rats, Revisited:

The complete data vector here is the number of rats of each species in each owl pellet. The incomplete data log-likelihood is:

$$\sum_{k \in \text{observations}} \log \left(\sum_i \pi_i f_i(\boldsymbol{W}_k) \right)$$

which is difficult to deal with because of the sum inside the logarithm. We represent the type of rat a skull came from with a $g \times n$ array of binary random variables $\mathcal{Z}$, where the l, j'th element of $\mathcal{Z}$ is z_{lj}. This element is one if the j'th skull is of type l, and zero otherwise.

The M-step: If we know $\mathcal{Z}$, then an estimate of the π_l is easy. A maximum likelihood estimate of the probability a rat of a particular type is eaten is given by the frequency with which those rats are eaten, i.e. by

$$\pi_l = \frac{\sum_{j=1}^n z_{lj}}{n}$$

(if you're not confident about this, you should check it, perhaps with chapter **??** in hand). Now we will never have $\mathcal{Z}$, but will have the expected value of $\mathcal{Z}$. The elements of this expected value will *not necessarily* be only 0 or 1 — they could take any value between zero or one. The correct intepretation of a value of, say, 0.2 is that this represents an observation that occured 0.2 times (i.e. rather less than once). Thus, in the likelihood, we raise the term corresponding to this data item to the 0.2'th power (in the same way that, if an observation occured three times, the term corresponding to the data item would appear as third power because three copies of the term would be multiplied together). This means that our maximum likelihood estimate obtained from $\overline{\mathcal{Z}}$ would be

$$\pi_l = \frac{\sum_{j=1}^{n} \overline{z}_{lj}}{n}$$

The E-step: Now for the E-step we want an expectation for z_{lj} given the $\boldsymbol{W}_k$'s. This expectation is:

$$\begin{aligned}\overline{z}_{lj} &= 1 \times P\left\{z_{lj} = 1 | \boldsymbol{W}_1 \ldots \boldsymbol{W}_n\right\} + 0 \times P\left\{z_{lj} = 0 | \boldsymbol{W}_1 \ldots \boldsymbol{W}_n\right\} \\ &= P\left\{z_{lj} = 1 | \boldsymbol{W}_j\right\} \\ &= \frac{P\left\{\boldsymbol{W}_j | z_{lj} = 1\right\} P\left\{z_{lj}\right\}}{P\left\{\boldsymbol{W}_j\right\}}\end{aligned}$$

Now we assume a uniform prior on z_{ij}, and deal with the question of $P\left\{\boldsymbol{W}_j\right\}$ by getting the probability to normalize to one, yielding

$$\overline{z}_{lj} = \frac{\pi_l f_l(\boldsymbol{W}_j)}{\sum_{i=1}^{n} \pi_i f_i(\boldsymbol{W}_j)}$$

Example: Image Segmentation, Revisited

This example works rather like the previous example. Assume there are a total of n pixels. The missing data forms an n by g array of indicator variables $\mathcal{I}$. In each row there is a single one, and all other values are zero — this indicates the blob from which each pixel's feature vector came.

The E-step: Now the expected value of the l, m'th entry of $\mathcal{I}$ is given by the probability that the l'th pixel comes from the m'th blob, using the same reasoning as for the owls and the rats. Assuming that the parameters are for the s'th iteration are $\Theta^{(s)}$, we have:

$$\overline{I}_{lm} = \frac{\alpha_m^{(s)} p_m(\boldsymbol{x}_l | \theta_l^{(s)})}{\sum_{k=1}^{K} \alpha_k^{(s)} p_k(\boldsymbol{x}_l | \theta_l^{(s)})}$$

(keeping in mind that $\alpha_m^{(s)}$ means the value of α_m on the s'th iteration!).

M-step: Once we have an expected value of $\mathcal{I}$, the rest is easy. We are essentially forming maximum likelihood estimates of Θ^{s+1}. Again, the expected value of the indicator variables is not in general going to be zero or one; instead, they will take some value in that range. This should be interpreted as an observation of that particular case that occurs with that frequency, meaning that the term in the likelihood corresponding to a particular indicator variable is raised to the power of the expected value. The calculation yields expressions for a weighted mean and weighted standard deviation that should be familiar:

$$\begin{aligned}
\alpha_m^{(s+1)} &= \frac{1}{r}\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)}) \\
\boldsymbol{\mu}_m^{(s+1)} &= \frac{\sum_{l=1}^{r} \boldsymbol{x}_l p(m|\boldsymbol{x}_l, \Theta^{(s)})}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})} \\
\Sigma_m^{s+1} &= \frac{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)}) \left\{(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})^T\right\}}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}
\end{aligned}$$

(again, keeping in mind that $\alpha_m^{(s)}$ means the value of α_m on the s'th iteration!).

Example: Line Fitting with EM

An EM line fitting algorithm follows the lines of the owls and rats example above; the missing data is an array of indicator variables $\mathcal{L}$ whose l, j'th element l_{lj} is one if point l is drawn from line j, zero otherwise. As in that example, the expected value is given by determining $P(l_{lj} = 1|\text{point } l, \text{line } j\text{'s parameters})$, and this probability is proportional to

$$\exp\left(-\frac{\text{distance from point } l \text{ to line } j^2}{2\sigma^2}\right)$$

for σ as above. The constant of proportionality is most easily determined from the fact that

$$\sum_i P(l_{ij} = 1|\text{point } i, \text{line } j\text{'s parameters}) = \sum_j P(l_{ij} = 1|\text{point } i, \text{line } j\text{'s parameters}) = 1$$

The maximisation follows the form of that for fitting a single line to a set of points, only now it must be done k times and the point coordinates are weighted by the value of $\overline{l_{lj}}$. Convergence can be tested by looking at the size of the change in the lines, or by looking at the sum of perpendicular distances of points from their lines (which operates as a log likelihood, see question **??**).

Both algorithms are inclined to get stuck in local minima. In one sense, these local minima are unavoidable — they follow from the assumption that the points are interchangeable (see figure 17.1). This can be dodged by noticing that the final configuration of either fitter is a deterministic function of its start point, and using carefully chosen start points. One strategy is to start in many different (randomly chosen) configurations, and sift through the results looking for the best fit. Another

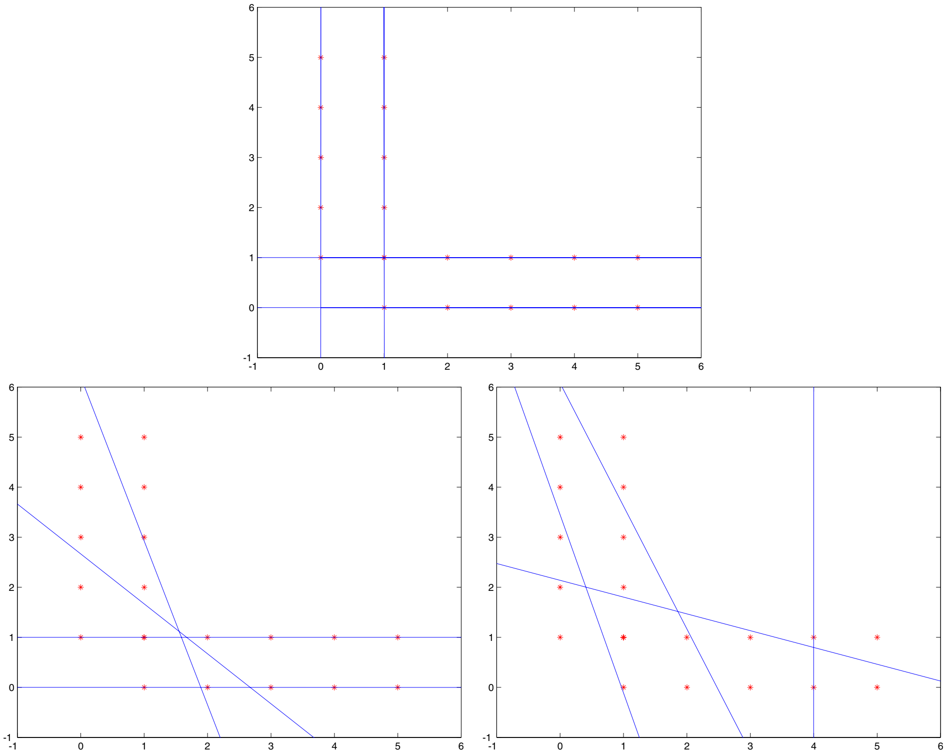


Figure 17.1. The top figure shows a good fit obtained using EM line fitting. The two bad examples in the bottom row were run with the right number of lines, but have converged to poor fits — which can be fairly good interpretations of the data, and are definitely local minima. This implementation adds a term to the mixture model that models the data point as arising uniformly and at random on the domain; a point that has a high probability of coming from this component has been identified as noise. Further examples of poor fits appear in figure 17.2.

is to preprocess the data using something like a Hough transform to guess good initial line fits. Neither is guaranteed. A cleaner approach is to notice that we are seldom, if ever, faced with a cloud of indistinguishable points and required to infer some structure on that cloud; usually, this is the result of posing a problem poorly. If points are not indistinguishable and have some form of linking structure, then a good start point should be much easier to choose.

A second difficulty to be aware of is that some points will have extremely small expected weights. This presents us with a numerical problem; it isn't clear what will happen if we regard small weights as being equivalent to zero (this isn't usually a wise thing to do). In turn, we may need to adopt a numerical representation which allows us to add many very small numbers and come up with a non-zero result.

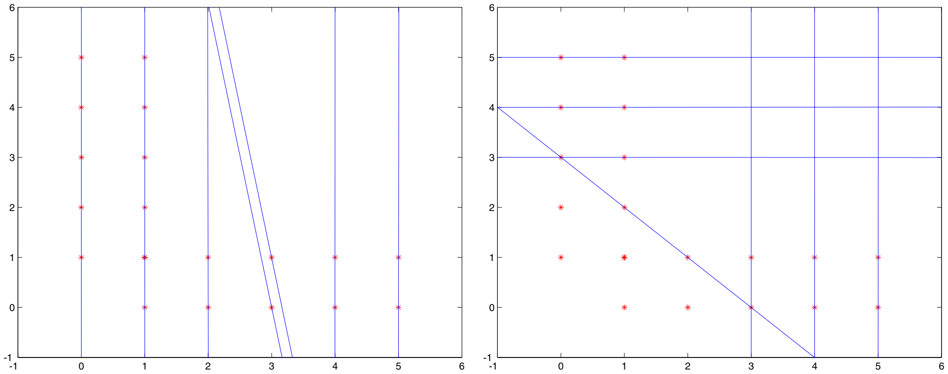


Figure 17.2. More poor fits to the data shown in figure 17.1; for these examples, we have tried to fit seven lines to this data set. Notice that these fits are fairly good interpretations of the data; they are local extrema of the likelihood. This implementation adds a term to the mixture model that models the data point as arising uniformly and at random on the domain; a point that has a high probability of coming from this component has been identified as noise. The fit on the bottom left has allocated some points to noise, and fits the others very well.

```
Choose $k$ lines (perhaps uniformly at random)
or choose $\overline{\mathcal{L}}$
Until convergence
  e-step:
    recompute $\overline{\mathcal{L}}$, from perpendicular distances
  m-step:
    refit lines using weights in $\overline{\mathcal{L}}$
```

Algorithm 17.1: *EM line fitting by weighting the allocation of points to each line, with the closest line getting the highest weight*

This issue is rather outside the scope of this book; you should not underestimate its nuisance value because we don't treat it in detail.

17.1.3 Colour and Texture Segmentation with EM

We have already done most of the work for this example in section 17.1. It remains to specify appropriate feature vectors, and discuss such matters as starting the EM algorithm. The results shown in figures 17.3-17.4 use three colour features — the coordinates of the pixel in L*a*b*, after the image has been smoothed — and three texture features — which use filter outputs to estimate local scale, anisotropy and contrast (figure 17.3); other features may well be more effective — and the position

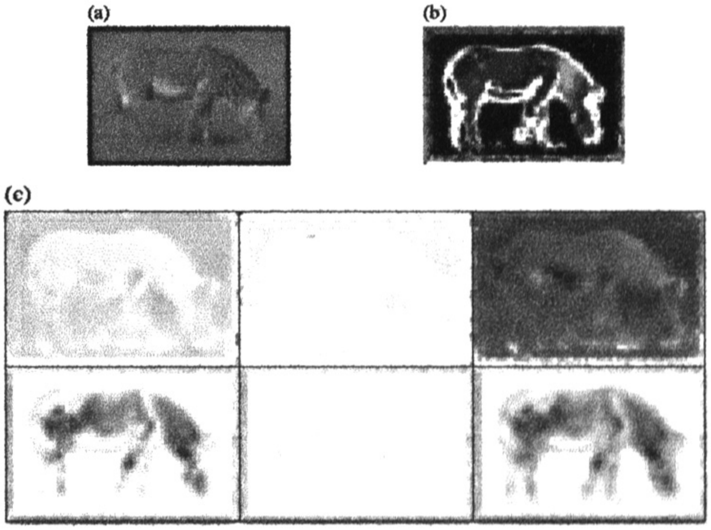


Figure 17.3. The image of the zebra in (a) gives local scale measurements shown in (b). These scale measurements essentially measure the scale of the change around a pixel; at edges, the scale is narrow, and in stripey regions it is broad, for example. The features that result are shown in (c); the top three images show the smoothed colour coordinates and the bottom three show the texture features (*ac*, *pc* and *c* — the scale and anisotropy features are weighted by contrast). *figure from Belongie et al, Color and Texture Based Image Segmentation Using EM and Its Application to Content Based Image Retrieval, ICCV98, p 5, in the fervent hope that permission will be granted*

of the pixel.

What should the segmenter report? One option is to choose for each pixel the value of m for which $p(m|\boldsymbol{x}_l, \Theta^s)$ is a maximum. Another is to report these probabilities, and build an inference process on top of them.

17.1.4 Motion Segmentation and EM

Missing data problems turn up all over computer vision. For example, motion sequences quite often consist of large regions which have quite similar motion internally. Let us assume for the moment that we have a very short sequence — two frames — and wish to determine the motion field at each point on the first frame. We will assume that the motion field comes from a mixture model. Recall that a general mixture model is a weighted sum of densities — the components do not have to have the Gaussian form used in section 17.1.1 (missing data, the EM algorithm and general mixture models turn up rather naturally together in vision applications).

A generative model for a pair of images would have the following form:

```
Choose a number of segments

Construct a set of support maps, one per segment,
containing one element per pixel.  These support maps
will contain the weight associating a pixel with a segment

Initialize the support maps by either:

    Estimating segment parameters from small
    blocks of pixels, and then computing weights
    using the E-step;

    Or randomly allocating values to the support maps.

Until convergence

    Update the support maps with an E-Step

    Update the segment parameters with an M-Step

end
```

Algorithm 17.2: *Colour and texture segmentation with EM*

- At each pixel in the first image, there is a motion vector connecting it to a pixel in the second image;
- there are a set of different parametric motion fields, each of which is given by a different probabilistic model;
- the overall motion is given by a mixture model, meaning that to determine the image motion at a pixel, we firstly determine which component the motion comes from, and then secondly draw a sample from this component.

This model encapsulates a a set of distinct, internally consistent motion fields — which might come from, say, a set of rigid objects at different depths and a moving camera (figure 17.5) — rather well.

Now assume that the motion fields have a parametric form, and that there are g different motion fields. Given a pair of images, we wish to determine (1) which motion field a pixel belongs to and (2) the parameter values for each field. All this should look a great deal like the first two examples, in that if we knew the first, the second would be easy, and if we knew the second, the first would be easy. This

```
For each pixel location $l$

    For each segment $m$

      Insert $\alpha_m^{(s)} p_m(\boldsymbol{x}_l|\theta_l^{(s)})$
      in pixel location $l$ in the support map $m$

end
    Add the support map values to obtain
    $\sum_{k=1}^{K} \alpha_k^{(s)} p_k(\boldsymbol{x}_l|\theta_l^{(s)})$
    and divide the value in location $l$ in each support map by this term

end
```

Algorithm 17.3: *Colour and texture segmentation with EM: - the E-step*

```
For each segment $m$

    Form new values of the segment parameters
    using the expressions:

$\alpha_m^{(s+1)} = \frac{1}{r} \sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})$

$\boldsymbol{\mu}_m^{(s+1)} = \frac{\sum_{l=1}^{r} \boldsymbol{x}_l p(m|\boldsymbol{x}_l, \Theta^{(s)})}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}$

$\Sigma_m^{s+1} = \frac{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)}) \{(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})(\boldsymbol{x}_l - \boldsymbol{\mu}_m^{(s)})^T\}}{\sum_{l=1}^{r} p(m|\boldsymbol{x}_l, \Theta^{(s)})}$

    Where $p(m|\boldsymbol{x}_l, \Theta_{(s)})$ is the value
    in the $m$'th support map for pixel location $l$

end
```

Algorithm 17.4: *Colour and texture segmentation with EM: - the M-step*

is again a missing data problem: the missing data is the motion field to which a pixel belongs, and the parameters are the parameters of each field and the mixing

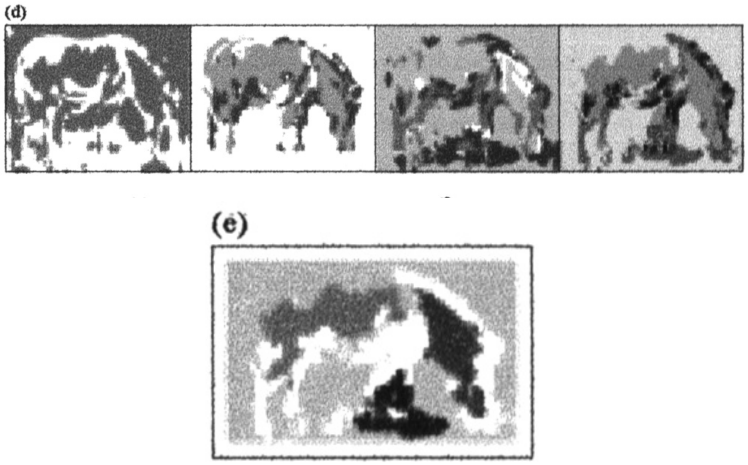


Figure 17.4. Each pixel of the zebra image of figure 17.3 is labelled with the value of m for which $p(m|\boldsymbol{x}_l, \Theta^s)$ is a maximum, to yield a segmentation. The images in (d) show the result of this process for $K = 2, 3, 4, 5$. Each image has K grey-level values corresponding to the segment indexes. *figure from Belongie et al, Color and Texture Based Image Segmentation Using EM and Its Application to Content Based Image Retrieval, ICCV98, p 5, in the fervent hope that permission will be granted*

Missing
Figure

Figure 17.5. A pair of frames from a sequence of a garden, with motion segments overlaid

weights.

Assume that the pixel at (u, v) in the first image belongs to the l'th motion field, with parameters θ_l. This means that this pixel has moved to $(u, v) + \boldsymbol{m}(u, v; \theta_l)$ in the second frame, and so that the intensity at these two pixels is the same up to measurement noise. We will write $I_1(u, v)$ for the image intensity of the first image at the u, v'th pixel, and so on. The missing data is the motion field to which the pixel belongs. We can represent this by an indicator variable $V_{uv,l}$ where

$$V_{uv,l} = \left\{ \begin{array}{c} 1, \text{ if the } u,\ v\text{'th pixel belongs to the } l\text{'th motion field} \\ 0, \text{ otherwise} \end{array} \right\}$$

We assume Gaussian noise with standard deviation σ in the image intensity values, so the complete data log-likelihood is

$$L(V, \Theta) = -\sum_{ij,l} V_{uv,l} \frac{(I_1(u,v) - I_2(u + m_1(u,v;\theta_l), v + m_2(u,v;\theta_l)))^2}{2\sigma^2} + C$$

where $\Theta = (\theta_1, \ldots, \theta_g)$. Setting up the EM algorithm from here on is straightforward. As above, the crucial issue is determining

$$P\left\{V_{uv,l} = 1 | I_1, I_2, \Theta\right\}$$

The more interesting question is the appropriate choice of parametric motion model. A common choice is an **affine motion model**, where

$$\left\{ \begin{array}{c} m_1 \\ m_2 \end{array} \right\} (i, j; \theta_l) = \left\{ \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right\} \left\{ \begin{array}{c} i \\ j \end{array} \right\} + \left\{ \begin{array}{c} a_{13} \\ a_{23} \end{array} \right\}$$

and $\theta_l = (a_{11}, \ldots, a_{23})$.

17.1.5 The Number of Components

At each stage of this section, we have assumed that the number of components in the mixture model is known. This is generally not the case in practice. Finding the number of components is, in essence, a model selection problem — we will search through the collection of models (where different models have different numbers of components) to determine which fits the data best. Of course, if we simply look at the likelihood, the more components in the model, the better the fit, so we need to apply a term of that penalizes the model for more components. This issue arises quite widely, and we deal with it in detail in section 17.3.

17.1.6 How Many Lines are There?

It isn't possible to do anything sensible about this problem without a model. After all, there could be one line for every pair of points, or no lines at all and a hyperactive noise process.

Incremental line fitters appear to solve this problem without a model — after all, one applies the incremental line fitter, and some collection of lines emerges. In fact, the series of tests that are applied to determine whether a line is present constitute a model. These tests are typically a test on the residual to determine whether edge points are associated with a particular line, and some test to ensure that lines have sufficient support. Because the model is not explicit, it can be difficult to understand its implications.

When there is an explicit model, this problem is a model selection problem (as in section **??**): does the evidence support a model with $r + 1$ lines better than a model with r lines? There are a variety of model selection methods that apply to this problem as well as to telling how many segments are in an image, etc. We review these methods in section 17.3.

17.2 Robustness

In this section, we discuss a very general difficulty through the lens of our model problem. In particular, all of the line-fitting methods we have described involve squared error terms. This can lead to very poor fits in practice, because a single wildly inappropriate data point can give errors that are dominate those due to many good data points; these errors can result in a substantial bias in the fitting process (figure 17.6). It appears to be very difficult to avoid such data points — usually called **outliers** — in practice. Errors in collecting or transcribing data points is one important source of outliers. Another common source is a problem with the model — perhaps some rare but important effect has been ignored, or the magnitude of an effect has been badly underestimated[1]. Vision problems are usually full of outliers.

One approach to this problem puts the model at fault: the model predicts these outliers occuring perhaps once in the lifetime in the universe, and they clearly occur much more often than that. The natural response is to improve the model, either by allowing an explicit outlier model (section 17.2.1) or by giving the noise "heavier tails" (section 17.2.2). Finally, we could search for points that appear to be good (section **??**).

17.2.1 Explicit Outliers

The line fitters we have described have difficulty with outliers because they encounter outliers with a frequency that is wildly underpredicted by the model. Outliers are often referred to as being "in the **tails**" of a probability distribution. In probability distributions like the normal distribution, there is a large collection of values with very small probability; these values are the tails of the distribution (probably because these values are where the distribution *tails off*). A natural mechanism for dealing with outliers is to modify the model so that the distribution has **heavier tails** (i.e. that there is more probability in the tails).

One way to do this is to construct an explicit model of outliers, which is usually quite easy to do. We form a weighted sum of the likelihood $P(\text{measurements}|\text{model})$ and a term for outliers $P(\text{outliers})$, to obtain:

$$(1-\lambda)P(\text{measurements}|\text{model}) + \lambda P(\text{outliers})$$

here $\lambda \in [0,1]$ models the frequency with which outliers occur, and $P(\text{outliers})$ is some probability model for outliers; failing anything better, it could be uniform over the possible range of the data.

---

[1] At least one author of a textbook on statistics claims to have two patents due to outliers [], p.

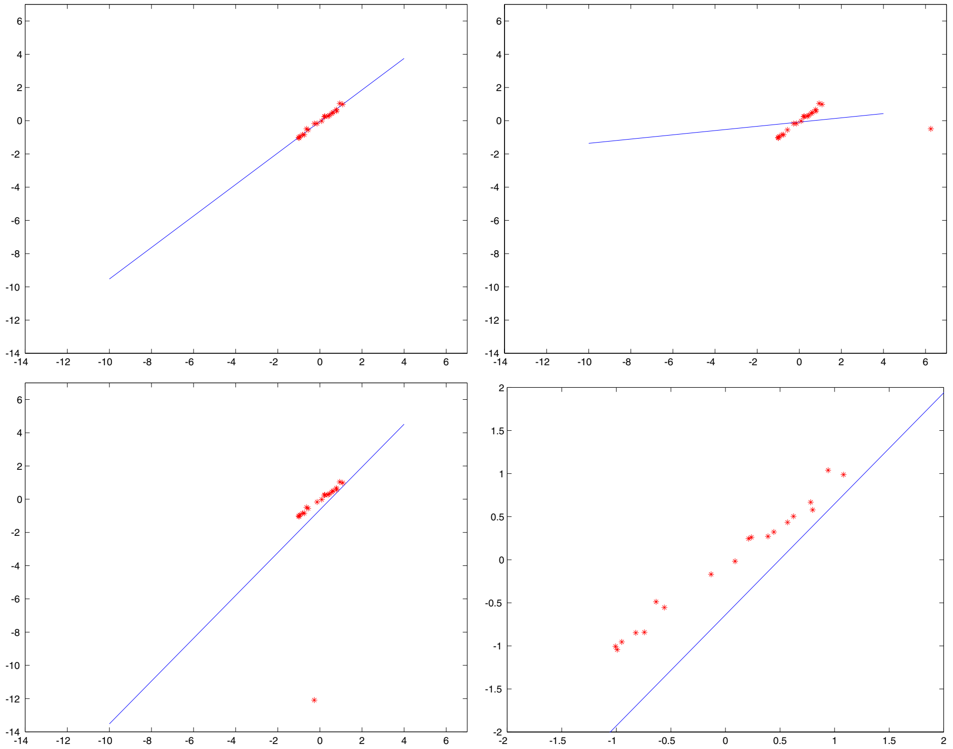


Figure 17.6. Least squares line fitting is hideously sensitive to outliers, both in x and y coordinates. At the top left, a good least-squares fit of a line to a set of points. Top-right shows the same set of points, but with the x-coordinate of one point corrupted. In this case, the slope of the fitted line has swung wildly. Bottom-left shows the same set of points, but with the y-coordinate of one point corrupted. In this particular case, the x-intercept has changed. These three figures are on the same set of axes for comparison, but this choice of axes does not clearly show how bad the fit is for the third case; Bottom-right shows a detail of this case — the line is clearly a very bad fit.

The natural way to deal with this model is to construct a variable that indicates which component generated each point. With this variable, we have a complete data likelihood function with an easy form. Of course, we don't *know* this variable, but this is a missing data problem, and we know how to proceed here using EM (you provide the details in the exercises!). The usual difficulties with EM occur here, too. In particular, it is easy to get trapped in local minima, and we may need to be careful about the numerical representation adopted for very small probabilities.

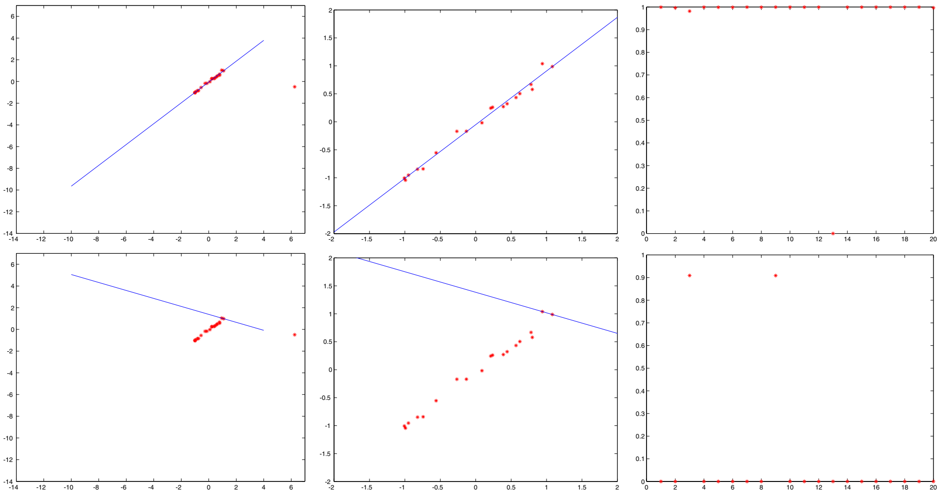


Figure 17.7. EM can be used to reject outliers; here we demonstrate a line fit to the second data set of figure 17.6. The top row shows the correct local minimum, and the bottom row shows another local minimum. The first column shows the line superimposed on the data points using the same axes as figure 17.6; the second column shows a detailed view of the line, indicating the region around the data points; and the third column shows a plot of the probability that a point comes from the line, rather than from the noise model, plotted against the index of the point. Notice that at the correct local minimum, all but one point is associated with the line, whereas at the incorrect local minimum, there are two points associated with the line and the others are allocated to noise.

17.2.2 M-estimators

The difficulty with modelling the source of outliers is that the model might be wrong. Generally, the best we can hope for from a probabilistic model of a process is that it is quite close to the right model. Assume that we are guaranteed that our model of a process is close to the right model — say, the distance between the density functions in some appropriate sense is less than ϵ. We can use this guarantee to reason about the design of estimation procedures for the parameters of the model. In particular we can choose an estimation procedure by assuming that nature is malicious and well-informed about statistics[2]. In this line of reasoning, we assess the goodness of an estimator by assuming that somewhere in the collection of processes close to our model is the real process, and it just happens to be the one that makes the estimator produce the worst possible estimates. The best estimator is the one that behaves best on the worst distribution close to the parametric model. This is a criterion which can be used to produce a wide variety of estimators.

---

[2] Generally, sound assumptions for any enterprise; the world is full of opportunities for painful and expensive lessons in practical statistics

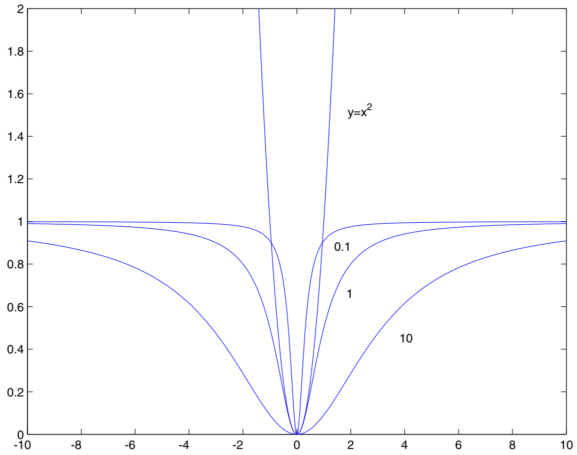


Figure 17.8. The function $\rho(x;\sigma) = x^2/(\sigma^2 + x^2)$, plotted for $\sigma^2 = 0.1$, 1 and 10, with a plot of $y = x^2$ for comparison. Replacing quadratic terms with ρ reduces the influence of outliers on a fit — a point which is several multiples of σ away from the fitted curve is going to have almost no effect on the coefficients of the fitted curve, because the value of ρ will be close to 1 and will change extremely slowly with the distance from the fitted curve.

An **M-estimator** estimates parameters by minimizing an expression of the form

$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma)$$

where θ are the parameters of the model being fitted and $r_i(\boldsymbol{x}_i, \theta)$ is the residual error of the model on the i'th data point. Generally, $\rho(u;\sigma)$ looks like u^2 for part of its range, and then flattens out. A common choice is

$$\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}$$

The parameter σ controls the point at which the function flattens out; we have plotted a variety of examples in figure 17.8. There are many other M-estimators available. Typically, they are discussed in terms of their **influence function**, which is defined as

$$\frac{\partial \rho}{\partial \theta}$$

This is natural, because our criterion is

$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma) \frac{\partial \rho}{\partial \theta} = 0$$

For the kind of problems we consider, we would expect a good influence function to be antisymmetric — there is no difference between a slight over prediction and a slight under prediction — and to tail off with large values — because we want to limit the influence of the outliers.

There are two tricky issues with using M-estimators. Firstly, the extremisation problem is non-linear and must be solved iteratively. The standard difficulties apply: there may be more than one local minimum; the method may diverge; and the behaviour of the method is likely to be quite dependent on the start point. A common strategy for dealing with this problem is to draw a subsample of the data set, fit to that subsample using least squares, and use this as a start point for the fitting process. We do this for a large number of different subsamples, enough to ensure that there is a very high probability that in that set there is at least one that consists entirely of good data points.

Secondly, as figures 17.9 and 17.10 indicate, the estimators require a sensible estimate of σ, which is often referred to as scale. Typically, the scale estimate is supplied at each iteration of the solution method; a popular estimate of scale is

$$\sigma^{(n)} = 1.4826 median_i |r_i^{(n)}(x_i;\theta^{(n-1)})|$$

```
For $s = 1$ to $s = k$
  draw a subset of $r$ distinct points, chosen uniformly at random

  Fit to this set of points using maximum likelihood
  (usually least squares) to obtain $\theta_s^0$

  estimate $\sigma_s^0$ using $\theta_s^0$

  Until convergence (usually $|\theta_s^n - \theta_s^{n-1}|$ is small):
    take a minimising step using $\theta_s^{n-1}$, $\sigma_s^{n-1}$
    to get $\theta_s^n$
    now compute $\sigma_s^n$

report the best fit of this set, using the median of the
residuals as a criterion
```

Algorithm 17.5: *Using an M-estimator to fit a probabilistic model*

An M-estimator can be thought of as a trick for ensuring that there is more probability in the tails than would otherwise occur with a quadratic error. The function that is minimised looks like distance for small values of $\boldsymbol{x}$ — thus, for valid data points the behaviour of the M-estimator should be rather like maximum

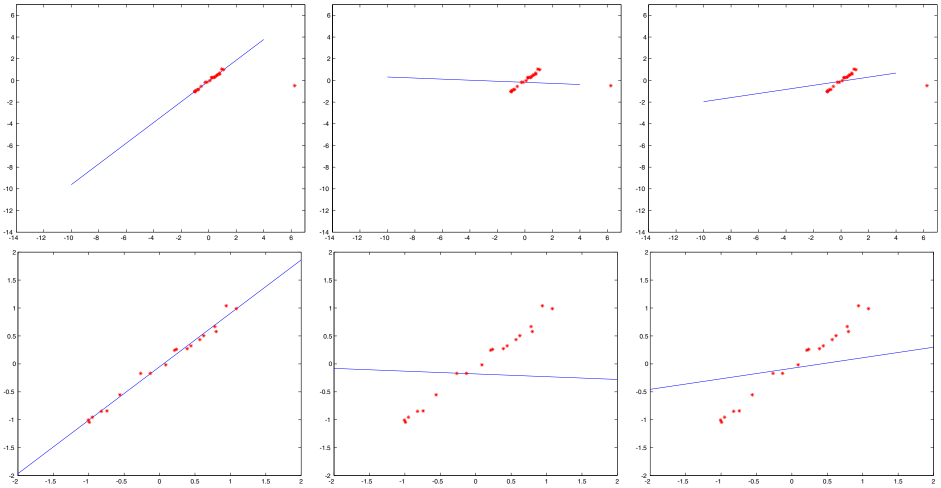


Figure 17.9. The top row shows lines fitted to the second dataset of figure 17.6 using a weighting function that de-emphasizes the contribution of distant points (the function ϕ of figure 17.8). On the left, μ has about the right value; the contribution of the outlier has been down-weighted, and the fit is good. In the center, the value of μ is too small, so that the fit is insensitive to the position of all the data points, meaning that its relationship to the data is obscure. On the right, the value of μ is too large, meaning that the outlier makes about the same contribution that it does in least-squares. The bottom row shows close-ups of the fitted line and the non-outlying data points, for the same cases.

likelihood — and like a constant for large values of $\boldsymbol{x}$ — meaning that a component of probability is given to the tails of the distribution. The strategy of the previous section can be seen as an M-estimator, but with the difficulty that the influence function is discontinuous, meaning that obtaining a minimum is tricky.

17.2.3 RANSAC

An alternative to modifying the generative model to have heavier tails is to search the collection of data points for good points. This is quite easily done by an iterative process: first, we choose a small subset of points and fit to that subset; then we see how many other points fit to the resulting object. We continue this process until we have a high probability of finding the structure we are looking for.

For example, assume that we have a data set that consists of about 50% outliers. If we draw pairs of points uniformly and at random, then about 1/4 of these pairs will consist entirely of good data points. We can identify these good pairs, by noticing that a large collection of other points will lie close to the line fitted to such a pair. Of course, a better estimate of the line could then be obtained by fitting a line to the points that lie close to our current line.

This approach leads to an algorithm — search for a random sample that leads

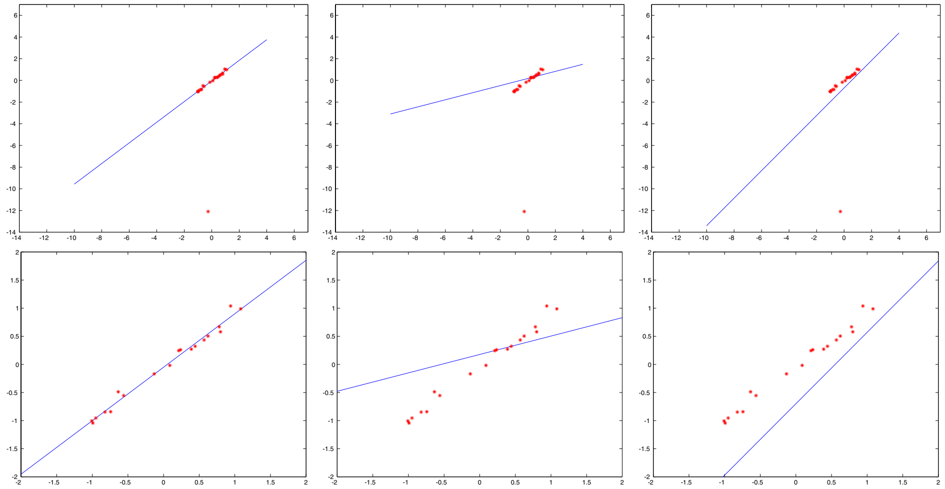


Figure 17.10. The top row shows lines fitted to the third dataset of figure 17.6 using a weighting function that de-emphasizes the contribution of distant points (the function ϕ of figure **??**). On the left, μ has about the right value; the contribution of the outlier has been down-weighted, and the fit is good. In the center, the value of μ is too small, so that the fit is insensitive to the position of all the data points, meaning that its relationship to the data is obscure. On the right, the value of μ is too large, meaning that the outlier makes about the same contribution that it does in least-squares. The bottom row shows close-ups of the fitted line and the non-outlying data points, for the same cases.

to a fit on which many of the data points agree. The algorithm is usually called `RANSAC`, for RANdom SAmple Consensus. To make this algorithm practical, we need to be able to choose three parameters.

How Many Samples are Necessary?

Our samples will consist of sets of points drawn uniformly and at random from the data set. Each sample will contain the minimum number of points required to fit the abstraction we wish to fit; for example, if we wish to fit lines, we will draw pairs of points; if we wish to fit circles, we will draw triples of points, etc. We assume that we need to draw n data points, and that w is the fraction of these points that are good (we will need only a reasonable estimate of this number). Now the expected value of the number of draws k required to get one point is given by

$$\begin{aligned} \mathrm{E}[k] &= 1P(\text{one good sample in one draw}) + 2P(\text{one good sample in two draws}) + \ldots \\ &= w^n + 2(1-w^n)w^n + 3(1-w^n)^2 w^n + \ldots \\ &= w^{-n} \end{aligned}$$

(where the last step takes a little manipulation of algebraic series). Now we would like to be fairly confident that we have seen a good sample, so we would wish to draw rather more than w^{-n} samples; a natural thing to do is to add a few standard deviations to this number (see section **??** for an inequality that suggests why this is the case). The standard deviation of k can be obtained as

$$SD(k) = \frac{\sqrt{1-w^n}}{w^n}$$

An alternative approach to this problem is to choose to look at a number of samples that guarantees a low probability z of seeing only bad samples. In this case, we have

$$(1-w^n)^k = (1-z)$$

which means that

$$k = \frac{\log(1-z)}{\log(1-w^n)}$$

How Far Away is Agreement?

We need to determine whether a point lies close to a line fitted to a sample. We will do this by determining the distance between the point and the fitted line, and testing that distance against a threshold d; if the distance is below the threshold, then the point lies close. In general, specifying this parameter is part of the modelling process. For example, when we fitted lines using maximum likelihood, there was a term σ in the model (which disappeared in the manipulations to find an maximum). This term gives the average size of deviations from the model being fitted.

In general, obtaining a value for this parameter is relatively simple. We generally need only an order of magnitude estimate, and the same value will apply to many different experiments. The parameter is often determined by trying a few values and seeing what happens; another approach is to look at a few characteristic data sets, fitting a line by eye, and estimating the average size of the deviations.

How Many Points Need to Agree?

Assume that we have fitted a line to some random sample of two data points. We need to know whether that line is good or not. We will do this by counting the number of points that lie within some distance of the line (the distance was determined in the previous section). In particular, assume that we know the probability that an outlier lies in this collection of points; write this probability as y. We should like to choose some number of points t such that y^t is small (say, less than 0.05).

There are two ways to proceed. One is to notice that $y \leq (1-w)$, and to choose t such that $(1-w)^t$ is small. Another is to get an estimate of y from some model of outliers — for example, if the points lie in a unit square, the outliers are uniform, and the distance threshold is d, then $y \leq 2\sqrt{2}d$.

These observations lead to algorithm 6, originally due to [**?**].

```
Determine $n$, $t$, $d$ and $k$ as above
Until there is a good fit or $k$ iterations have occurred
    draw a sample of $n$ points from the data
    uniformly and at random

    fit to that set of $n$ points

    for each data point outside the sample

      test the distance from the point to the line
      against $t$; if the distance from the point to the line
      is less than $t$, the point is close

    end
    if there are $d$ or more points close to the line
    then there is a good fit.  Refit the line using all
    these points, and terminate
end
```

Algorithm 17.6: *RANSAC: fitting using random sample consensus*

17.3 How Many are There?

In all the discussion above, we have assumed that the number of components (lines, etc.) was known. This is hardly ever the case in practice. Instead, we need to refer to some statistical procedure to infer the number of components from the data and the model. This is another instance of model selection (which we discussed in section 21). We adopt a strategy of searching over the number of components — perhaps restricting the search to a reasonable range for reasons of computational complexity — and evaluating a score for each case. The best score gives the number of components.

Generally, the value of the negative log-likelihood is a poor guide to the number of components because, in general, a model with more parameters will fit a dataset better than a model with fewer parameters. This means that simply minimizing the negative log-likelihood as a function of the number of components will tend to lead to too many components. For example, we can fit a set of lines extremely accurately by passing a line through each pair of points — there may be a lot of lines, but the fitting error is zero. We resolve this difficulty by adding a term that *increases* with the number of components — this penalty compensates for the decrease in negative log-likelihood caused by the increasing number of parameters.

It is important to understand that there is no *canonical* model selection process. Instead, we can choose from a variety of techniques, each of which uses a different discount corresponding to a different extremality principle (and different approximations to these criteria!).

17.3.1 Basic Ideas

Model selection is a general problem in fitting parametric models. The problem can be set up as follows: there is a data set, which is a sample from a parametric model which is itself a member of a family of models. We wish to determine (1) which model the data set was drawn from and (2) what the parameters of that model were. A proper choice of the parameters will predict future samples from the model — a **test set** — *as well as* the data set (which is often called the **training set**); unfortunately, these future samples are not available. Furthermore, the estimate of the model's parameters obtained using the data set is likely to be biased, because the parameters chosen ensure that the model fits the *training set* — rather than the entire set of possible data — optimally. The effect is known as **selection bias**. The training set is a subset of the entire set of data that could have been drawn from the model, and represents the model exactly only if it is infinitely large. This is why the negative log-likelihood is a poor guide to the choice of model: the fit looks better, because it is increasingly biased.

The correct penalty to use comes from the **deviance**, given by

> twice (log-likelihood of the best model minus log-likelihood of the current model).

(from Ripley, [], p. 348); the best model should be the true model. Ideally, the deviance would be zero; the argument above suggests that the deviance on a training set will be larger than the deviance on a test set. A natural penalty to use is the difference between these deviances averaged over both test and training sets. This penalty is applied to twice the log-likelihood of the fit — the factor of two appears for reasons we cannot explain, but has no effect in practice. Let us write the best choice of parameters as Θ^* and the log-likelihood of the fit to the data set as $L(\boldsymbol{x}; \Theta^*)$.

17.3.2 AIC — An Information Criterion

Akaike proposed a penalty, widely called **AIC**[3] which leads to minimizing

$$-2L(\boldsymbol{x}; \Theta^*) + 2p$$

where p is the number of free parameters. There is a collection of statistical debate about the AIC (which could be followed up in [**?**; **?**; **?**]). The first main point is that it lacks a term in the *number* of data points. This is suspicious, because the deviance between a fitted model and the real model should go down as the number of data points goes up. Secondly, there is a body of experience that the AIC tends to **overfit** — that is, to choose a model with too many parameters which fits the training set well but doesn't perform as well on test sets (e.g. []).

---

[3] For "An information criterion," *not* "Akaike information criterion", []

17.3.3 Bayesian methods and Schwartz' BIC

We discussed Bayesian model selection in section **??**, coming up with the expression:

$$\begin{aligned} P(\text{model}|\text{data}) &= \frac{P(\text{data}|\text{model})}{P(\text{data})} \\ &= \frac{\int P(\text{data}|\text{model, parameters})P(\text{parameters})d\{\text{parameters}\}}{P(\text{data})} \\ &\propto \int P(\text{data}|\text{model, parameters})P(\text{parameters})d\{\text{parameters}\} \end{aligned}$$

A number of possibilities now present themselves. We could give the posterior over the models, or choose the MAP model. The first difficulty is that we need to specify a prior over the models. For example, in the case of EM based segmentation, we would need to specify a prior on the number of segments. The second difficulty is that we need to compute the integral over the parameters. This could be done using a sampling method ([], or section **??**). An alternative is to construct some form of approximation to the integral, which yields another form of the penalty term.

For simplicity, let us write $\mathcal{D}$ for the data, $\mathcal{M}$ for the model, and θ for the parameters. The extent to which data support model i over model j is proportional to the **Bayes factor**

$$\frac{P(\mathcal{D}|\mathcal{M}_i)}{P(\mathcal{D}|\mathcal{M}_j)} = \frac{\int P(\mathcal{D}|\mathcal{M}_i,\theta)P(\theta)d\theta}{\int P(\mathcal{D}|\mathcal{M}_j,\theta)P(\theta)d\theta} = \frac{P(\mathcal{M}_i|\mathcal{D})}{P(\mathcal{M}_j|\mathcal{D})}\frac{P(\mathcal{M}_j)}{P(\mathcal{M}_i)} \propto \frac{P(\mathcal{M}_i|\mathcal{D})}{P(\mathcal{M}_j|\mathcal{D})}$$

Assume that $P(\mathcal{D},\theta|\mathcal{M})$ looks roughly normal — i.e. has a single mode, roughly elliptical level curves, and doesn't die off too fast. Now write the value of θ at the mode as θ^*. The **Hessian** at the mode is a matrix $\mathcal{H}$ whose i, j'th element is

$$\frac{\partial^2 \log p(\boldsymbol{x}_i;\Theta)}{\partial\theta_i\partial\theta_j}$$

evaluated at $\theta = \theta^*$.

If we take a Taylor series approximation to the log of $P(\mathcal{D},\theta|\mathcal{M})$ as a function of θ at the mode ($\theta = \theta^*$), we get

$$\begin{aligned} \log P(\mathcal{D},\theta|\mathcal{M}) &= L(\mathcal{D};\theta) + \log p(\theta) + \text{constant} \\ &= \Phi(\theta) \\ &= \Phi(\theta^*) + (\theta-\theta^*)^T\mathcal{H}(\theta-\theta^*) + O((\theta-\theta^*)^3) \\ &\approx \Phi(\theta^*) + (\theta-\theta^*)^T\mathcal{H}(\theta-\theta^*) \end{aligned}$$

(the linear term is missing because θ^* is the mode). Now

$$P(\mathcal{D}|\mathcal{M}) = \int P(\mathcal{D},\theta|\mathcal{M})d\theta$$

$$
\begin{aligned}
&= \int \exp \Phi(\theta) d\theta \\
&\approx \exp \Phi(\theta^*) \int \exp(\theta - \theta^*)^T \mathcal{H}(\theta - \theta^*) d\theta \\
&= \{\exp \Phi(\theta^*)\} (2\pi)^{\frac{p}{2}} |\mathcal{H}^{-1}|^{\frac{1}{2}}
\end{aligned}
$$

All this implies that

$$
\log p(\mathcal{D}|\mathcal{M}) \approx L(\mathcal{D};\theta^*) + \log p(\theta^*) + \frac{p}{2}\log(2\pi) + \frac{1}{2}\log|\mathcal{H}^{-1}|
$$

Given a reasonable optimisation process and a bit of luck, we could find θ^* and $\mathcal{H}$, and evaluate this expression and so the Bayes factor. This analysis offers a criterion

$$
-L(\mathcal{D};\theta^*) - \left\{\log p(\theta^*) + \frac{p}{2}\log(2\pi) + \frac{1}{2}\log|\mathcal{H}^{-1}|\right\}
$$

(the signs are to allow comparison with the AIC in section 17.3.2). This might be difficult to evaluate. A series of approximations starting here leads to a criterion

$$
-L(\mathcal{D};\theta^*) + \frac{p}{2}\log N
$$

called the **Bayes information criterion** or BIC.

17.3.4 Description Length

Models can be selected by criteria that are not intrinsically statistical; after all, we are selecting the model and we can say why we want to select it. A criterion that is somewhat natural is to choose the model that encodes the data set most crisply. This **minimum description length** criterion chooses the model that allows the most efficient transmission of the data set. To transmit the data set, one codes and transmits the model parameters, and then codes and transmits the data given the model parameters. If the data fits the model very poorly, then this latter term is large, because one has to code a noise-like signal.

A derivation of the criterion used in practice is rather beyond our needs. The details appear in [**?**]; similar ideas appear in [**?**; **?**]. Surprisingly, the BIC emerges from this analysis, yielding

$$
-L(\mathcal{D};\theta^*) + \frac{p}{2}\log N
$$

17.3.5 Other Methods for Estimating Deviance

The key difficulty in model selection is that we should be using a quantity we can't measure — the model's ability to predict data not in the training set. Given a sufficiently large training set, we could split the training set into two components, use one to fit the model and the other the test the fit. This is an approach known as **cross-validation**.

We can use cross-validation to determine the number of components in a model by splitting the data set, fitting a variety of different models to one side of the split, and then choosing the model that performs best on the other side. We expect this process to estimate the number of components, because a model that has too many parameters will fit the one data set well, but fit the other badly.

Using a single choice of a split into two components introduces a different form of selection bias, and the safest thing to do is to average the estimate over all such splits. This becomes unwieldy if the test set is large, because the number of splits is huge. The most usual version is **leave-one-out cross-validation**. In this approach we fit a model to each set of $N-1$ of the training set, compute the error on the remaining data point, and sum these errors to obtain an estimate of the model error. The model that minimizes this estimate is then chosen.

To our knowledge, this approach — which is standard for model selection in other kinds of problems — has not been used in fitting applications. It is certainly appropriate for estimating the number of components. First, assume that we compute the model error for a model with too few components to describe the image accurately. In this case, the model error will be large, because for many pixels the model will be insufficiently flexible to describe the pixel that was left out. Similarly, if we use too many components, the model will predict the left out pixel rather poorly.

17.4 Discussion

The topic of priors over models attracts considerable rhetoric. Objectors complain that this prior is arbitrary, and can be important. Supporters retort that working from such criteria as AIC or likelihood-ratios, etc., itself involves an implicit choice of prior, which might be a perverse choice for some cases.

Assignments

Exercises

- Write out the EM line fitting algorithm in detail, using the sketch in the text, the description of the owls and mice problem, and the model of section **??**. In particular, why does the E-step have the form it does? and the M-step?

- Derive the expressions of section 17.1.3 for segmentation. One possible modification is to use the new mean in the estimate of the covariance matrices. Perform an experiment to determine whether this makes any difference in practice.

- Derive the expressions for EM for motion.

- Describe using leave-one-out cross-validation for selecting the number of segments

Programming Assignments

- Build an EM segmenter that uses colour, position (ideally, use texture too) to segment images; use a model selection term to determine how many segments there should be. How significant a phenomenon is the effect of local minima?
- Build an EM line fitter that works for a fixed number of lines. Investigate the effects of local minima. One way to avoid being distracted by local minima is to start from many different start points, and then look at the best fit obtained from that set. How successful is this? how many local minima do you have to search to obtain a good fit for a typical data set? Can you improve things using a Hough transform?
- Expand your EM line fitter to incorporate a model selection term, so that the fitter can determine how many lines fit a dataset. Compare the choice of AIC and BIC.
- Insert a noise term in your EM line fitter, so that it is able to perform robust fits. What is the effect on the number of local minima? Notice that, if there is a low probability of a point arising from noise, most points will be allocated to lines, but the fits will often be quite poor; if there is a high probability of a point arising from noise, points will be allocated to lines only if they fit well. What is the effect of this parameter on the number of local minima?
- Construct a RANSAC fitter that can fit an arbitrary (but known) number of lines to a given data set. What is involved in extending your fitter to determine the best number of lines?

Chapter 18

TRACKING

Tracking is the problem of generating an inference about the motion of an object given a sequence of images. Good solutions to this problem have a variety of applications:

- **Motion Capture:** if we can track a moving person accurately, then we can make an accurate record of their motions. Once we have this record, we can use it to drive a rendering process; for example, we might control a cartoon character, thousands of virtual extras in a crowd scene, or a virtual stunt avatar. Furthermore, we could modify the motion record to obtain slightly different motions. This means that a single performer can produce sequences they wouldn't want to do in person.

- **Recognition From Motion:** the motion of objects is quite characteristic. We may be able to determine the identity of the object from its motion; we should be able to tell what it's doing.

- **Surveillance:** knowing what objects are doing can be very useful. For example, different kinds of trucks should move in different, fixed patterns in an airport; if they do not, then something is going very wrong. Similarly, there are combinations of places and patterns of motions that should never occur (no truck should ever stop on an active runway, say). It could be helpful to have a computer system that can monitor activities and give a warning if it detects a problem case.

- **Targeting:** a significant fraction of the tracking literature is oriented towards (a) deciding what to shoot and (b) hitting it. Typically, this literature describes tracking using radar or infra-red signals (rather than vision), but the basic issues are the same — what do we infer about an object's future position from a sequence of measurements? (i.e. where should we aim?)

In typical tracking problems, we have a model for the object's motion, and some set of measurements from a sequence of images. These measurements could be the position of some image points, the position and moments of some image regions, or pretty much anything else. They are not guaranteed to be relevant, in the sense that some could come from the object of interest and some might come from other objects, or from noise.

18.1 Tracking as an Abstract Inference Problem

Much of this chapter will deal with the algorithmics of tracking. In particular, we will see tracking as a probabilistic inference problem. The key technical difficulty is maintaining an accurate representation of the posterior on object position given measurements, and doing so efficiently.

We model the object as having some internal state; the state of the object at the i'th frame is typically written as $\boldsymbol{X}_i$. The capital letters indicate that this is a random variable — when we want to talk about a particular value that this variable takes, we will use small letters. The measurements obtained in the i'th frame are values of a random variable $\boldsymbol{Y}_i$; we shall write $\boldsymbol{y}_i$ for the value of a measurement, and, on occasion, we shall write $\boldsymbol{Y}_i = \boldsymbol{y}_i$ for emphasis. There are three main problems:

- **Prediction:** we have seen $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}$ — what state does this set of measurements predict for the i'th frame? to solve this problem, we need to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_{i-1} = \boldsymbol{y}_{i-1})$.
- **Data association:** Some of the measurements obtained from the i-th frame may tell us about the object's state. Typically, we use $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_{i-1} = \boldsymbol{y}_{i-1})$ to identify these measurements.
- **Correction:** now that we have $\boldsymbol{y}_i$ — the relevant measurements — we need to compute a representation of $P(\boldsymbol{X}_i|\boldsymbol{Y}_0 = \boldsymbol{y}_0, \ldots, \boldsymbol{Y}_i = \boldsymbol{y}_i)$.

18.1.1 Independence Assumptions

Tracking is very difficult without the following assumptions:

- **Only the immediate past matters:** formally, we require

  $$P(\boldsymbol{X}_i|\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1}) = P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})$$

  This assumption hugely simplifies the design of algorithms, as we shall see; furthermore, it isn't terribly restrictive if we're clever about interpreting $\boldsymbol{X}_i$ as we shall show in the next section.

- **Measurements depend only on the current state:** we assume that $\boldsymbol{Y}_i$ is conditionally independent of all other measurements given $\boldsymbol{X}_i$. This means that

  $$P(\boldsymbol{Y}_i, \boldsymbol{Y}_j, \ldots \boldsymbol{Y}_k|\boldsymbol{X}_i) = P(\boldsymbol{Y}_i|\boldsymbol{X}_i)P(\boldsymbol{Y}_j, \ldots, \boldsymbol{Y}_k|\boldsymbol{X}_i)$$

  Again, this isn't a particularly restrictive or controversial assumption, but it yields important simplifications.

These assumptions mean that a tracking problem has the structure of inference on a hidden Markov model (where both state and measurements may be on a continuous domain). You should compare this chapter with section **??**, which the use of hidden Markov models in recognition.

18.1.2 Tracking as Inference

We shall proceed inductively. Firstly, we assume that we have $P(\boldsymbol{X}_0)$, which is our "prediction" in the absence of any evidence. Now correcting this is easy: when we obtain the value of $\boldsymbol{Y}_0$ — which is $\boldsymbol{y}_0$ — we have that

$$\begin{aligned}
P(\boldsymbol{X}_0|\boldsymbol{Y}_0=\boldsymbol{y}_0) &= \frac{P(\boldsymbol{y}_0|\boldsymbol{X}_0)P(\boldsymbol{X}_0)}{P(\boldsymbol{y}_0)} \\
&= \frac{P(\boldsymbol{y}_0|\boldsymbol{X}_0)P(\boldsymbol{X}_0)}{\int P(\boldsymbol{y}_0|\boldsymbol{X}_0)P(\boldsymbol{X}_0)d\boldsymbol{X}_0} \\
&\propto P(\boldsymbol{y}_0|\boldsymbol{X}_0)P(\boldsymbol{X}_0)
\end{aligned}$$

All this is just Bayes rule, and we either compute or ignore the constant of proportionality depending on what we need. Now assume we have a representation of $P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})$.

Prediction

Prediction involves representing

$$P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})$$

Our independence assumptions make it possible to write

$$\begin{aligned}
P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1}) &= \int P(\boldsymbol{X}_i,\boldsymbol{X}_{i-1}|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})d\boldsymbol{X}_{i-1} \\
&= \int P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1},\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})d\boldsymbol{X}_{i-1} \\
&= \int P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_{i-1})d\boldsymbol{X}_{i-1}
\end{aligned}$$

Correction

Correction involves obtaining a representation of

$$P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)$$

Our independence assumptions make it possible to write

$$P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i) = \frac{P(\boldsymbol{X}_i,\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}{P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}$$

$$
\begin{aligned}
&= \frac{P(\boldsymbol{y}_i|\boldsymbol{X}_i, \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})P(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})}{P(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)} \\
&= P(\boldsymbol{y}_i|\boldsymbol{X}_i)P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})\frac{P(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})}{P(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)} \\
&= \frac{P(\boldsymbol{y}_i|\boldsymbol{X}_i)P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})}{\int P(\boldsymbol{y}_i|\boldsymbol{X}_i)P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})d\boldsymbol{X}_i}
\end{aligned}
$$

18.1.3 Overview

The key algorithmic issue involves finding a representation of the relevant probability densities that (a) is sufficiently accurate for our purposes and (b) allows these two crucial sums to be done quickly and easily. The simplest case occurs when the dynamics are linear, the measurement model is linear, and the noise models are Gaussian (section 18.2). Non-linearities introduce a host of unpleasant problems (section 18.3) and we discuss some current methods for handling them (section 18.4; the appendix gives another method that is unreliable but occasionally useful). We discuss data association in section 18.5, and show some examples of tracking systems in action in section 20.3.

18.2 Linear Dynamic Models and the Kalman Filter

There are good relations between linear transformations and Gaussian probability densities. The practical consequence is that, if we restrict attention to linear dynamic models and linear measurement models, both with additive Gaussian noise, all the densities we are interested in will be Gaussians. Furthermore, the question of solving the various integrals we encounter can usually be avoided by tricks that allow us to determine directly *which* Gaussian we are dealing with.

18.2.1 Linear Dynamic Models

In the simplest possible dynamic model, the state is advanced by multiplying it by some known matrix (which may depend on the frame), and then adding a normal random variable of zero mean, and known covariance. Similarly, the measurement is obtained by multiplying the state by some matrix (which may depend on the frame), and then adding a normal random variable of zero mean and known covariance. We use the notation

$$\boldsymbol{x} \sim N(\boldsymbol{\mu}, \Sigma)$$

to mean that $\boldsymbol{x}$ is the value of a random variable with a normal probability distribution with mean $\boldsymbol{\mu}$ and covariance Σ; notice that this means that, if $\boldsymbol{x}$ is one-dimensional — we'd write $x \sim N(\mu, v)$ — that its standard deviation is $\sqrt{v}$. We can write our dynamic model as

$$x_i \sim N(\mathcal{D}_i \boldsymbol{x}_{i-1}; \Sigma_{d_i})$$

$$y_i \sim N(\mathcal{M}_i \boldsymbol{x}_i; \Sigma_{m_i})$$

Notice that the covariances could be different from frame to frame, as could the matrices. While this model appears very limited, it is in fact extremely powerful; we show how to model some common situations below.

Drifting Points

Let us assume that $\boldsymbol{x}$ encodes the position of a point. If $\boldsymbol{D}_i = Id$, then the point is moving under random walk — its new position is its old position, plus some Gaussian noise term. This form of dynamics isn't obviously useful, because it appears that we are tracking stationary objects. It is quite commonly used for objects for which no better dynamic model is known — we assume that the random component is quite large, and hope we can get away with it.

This model also illustrates aspects of the **measurement matrix** $\mathcal{M}$. The most important thing to keep in mind is that we don't need to measure every aspect of the state of the point at every step. For example, assume that the point is in 3D: now if $\mathcal{M}_{3k} = (0, 0, 1)$, $\mathcal{M}_{3k+1} = (0, 1, 0)$ and $\mathcal{M}_{3k+2} = (1, 0, 0)$, then at every third frame we measure, respectively, the z, y, or x position of the point. Notice that we can still expect to be able to track the point, even though we measure only one component of its position at a given frame. If we have sufficient measurements, we can reconstruct the state — the state is **observable**. We explore observability in the exercises.

Constant Velocity

Assume that the vector $\boldsymbol{p}$ gives the position and $\boldsymbol{v}$ the velocity of a point moving with constant velocity. In this case, $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + (\Delta t)\boldsymbol{v}_{i-1}$ and $\boldsymbol{v}_i = \boldsymbol{v}_{i-1}$. This means that we can stack the position and velocity into a single state vector, and our model applies. In particular,

$$\boldsymbol{x} = \left\{ \begin{array}{c} \boldsymbol{p} \\ \boldsymbol{v} \end{array} \right\}$$

and

$$\mathcal{D}_i = \left\{ \begin{array}{cc} Id & (\Delta t) Id \\ 0 & Id \end{array} \right\}$$

Notice that, again, we don't have to observe the whole state vector to make a useful measurement. For example, in many cases we would expect that

$$\mathcal{M}_i = \left\{ \begin{array}{cc} Id & 0 \end{array} \right\}$$

i.e. that we see only the position of the point. Because we know that it's moving with constant velocity — that's the model — we expect that we could use these measurements to estimate the whole state vector rather well.

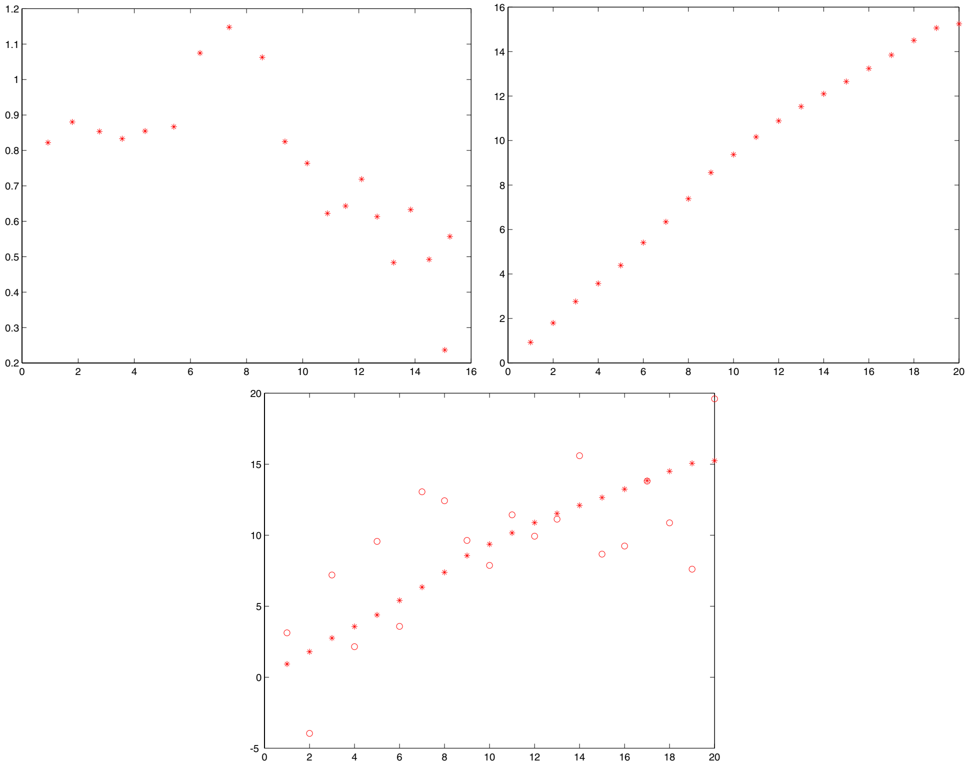


Figure 18.1. A constant velocity dynamic model, for a point on the line. In this case, the state space is two dimensional — one coordinate for position, one for velocity. The figure on the **top left** shows a plot of the state; each asterisk is a different state. Notice that the vertical axis (velocity) shows some small change, compared with the horizontal axis. This small change is generated only by the random component of the model, so that the velocity is constant up to a random change. The figure on the **top right** shows the first component of state (which is position) plotted against the time axis. Notice we have something that is moving with roughly constant velocity. The figure on the **bottom** overlays the measurements (the circles) on this plot. We are assuming that the measurements are of position only, and are quite poor; as we shall see, this doesn't significantly affect our ability to track.

Constant Acceleration

Assume that the vector $\boldsymbol{p}$ gives the position, vector $\boldsymbol{v}$ the velocity and vector $\boldsymbol{a}$ the acceleration of a point moving with constant acceleration. In this case, $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + (\Delta t)\boldsymbol{v}_{i-1}$, $\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + (\Delta t)\boldsymbol{a}_{i-1}$ and $\boldsymbol{a}_i = \boldsymbol{a}_{i-1}$. Again, we can stack the position, velocity and acceleration into a single state vector, and our model applies.

In particular,

$$\boldsymbol{x} = \left\{ \begin{array}{c} \boldsymbol{p} \\ \boldsymbol{v} \\ \boldsymbol{a} \end{array} \right\}$$

and

$$\mathcal{D}_i = \left\{ \begin{array}{ccc} Id & (\Delta t)Id & 0 \\ 0 & Id & (\Delta t)Id \\ 0 & 0 & Id \end{array} \right\}$$

Notice that, again, we don't have to observe the whole state vector to make a useful measurement. For example, in many cases we would expect that

$$\mathcal{M}_i = \left\{ \begin{array}{ccc} Id & 0 & 0 \end{array} \right\}$$

i.e. that we see only the position of the point. Because we know that it's moving with constant acceleration — that's the model — we expect that we could use these measurements to estimate the whole state vector rather well.

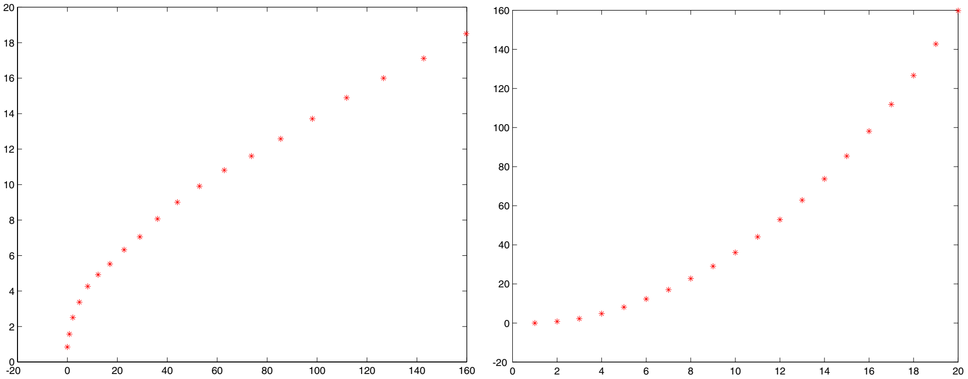


Figure 18.2. This figure illustrates a constant acceleration model for a point moving on the line. On the **left**, we show a plot of the first two components of state — the position on the x-axis and the velocity on the y-axis. In this case, we expect the plot to look like (t^2, t), which it does. On the **right**, we show a plot of the position against time — note that the point is moving away from its start position increasingly quickly.

Periodic Motion

Assume we have a point, moving on a line with a periodic movement. Typically, its position p satisfies a differential equation like

$$\frac{d^2p}{dt^2} = -p$$

This can be turned into a first order linear differential equation by writing the velocity as v, and stacking position and velocity into a vector $\boldsymbol{u} = (p, v)$; we then have

$$\frac{d\boldsymbol{u}}{dt} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \boldsymbol{u} = \mathcal{S}\boldsymbol{u}$$

Now assume we are integrating this equation with a forward Euler method, where the steplength is Δt; we have

$$\begin{aligned} \boldsymbol{u}_i &= \boldsymbol{u}_{i-1} + \Delta t \frac{d\boldsymbol{u}}{dt} \\ &= \boldsymbol{u}_{i-1} + \Delta t \mathcal{S} \boldsymbol{u}_{i-1} \\ &= \begin{pmatrix} 1 & \Delta t \\ -\Delta t & 1 \end{pmatrix} \boldsymbol{u}_{i-1} \end{aligned}$$

We can either use this as a state equation, or we can use a different integrator. If we used a different integrator, we might have some expression in $\boldsymbol{u}_{i-1}, \ldots, \boldsymbol{u}_{i-n}$ — we would need to stack $\boldsymbol{u}_{i-1}, \ldots, \boldsymbol{u}_{i-n}$ into a state vector and arrange the matrix appropriately (see the exercises). This method works for points on the plane, in 3D, etc. as well (again, see the exercises).

Higher Order Models

Another way to look at a constant velocity model is that we have augmented the state vector to get around the requirement that $P(\boldsymbol{x}_i|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}) = P(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})$. We could write a constant velocity model in terms of point position alone, as long as we were willing to use the position of the $i-2$'th point as well as that of the $i-1$'th point. In particular, writing position as $\boldsymbol{p}$, we would have

$$P(\boldsymbol{p}_i|\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{i-1}) = N(\boldsymbol{p}_{i-1} + (\boldsymbol{p}_{i-1} - \boldsymbol{p}_{i-2}), \Sigma_{d_i})$$

This model assumes that the difference between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i-1}$ is the same as the difference between $\boldsymbol{p}_{i-1}$ and $\boldsymbol{p}_{i-2}$ — i.e. that the velocity is constant, up to the random element. A similar remark applies to the constant acceleration model, which is now in terms of $\boldsymbol{p}_{i-1}$, $\boldsymbol{p}_{i-2}$ and $\boldsymbol{p}_{i-3}$.

We augmented the position vector with the velocity vector (which represents $\boldsymbol{p}_{i-1} - \boldsymbol{p}_{i-2}$) to get the state vector for a constant velocity model; similarly, we augmented the position vector with the velocity vector and the acceleration vector (which represents $(\boldsymbol{p}_{i-1} - \boldsymbol{p}_{i-2}) - (\boldsymbol{p}_{i-2} - \boldsymbol{p}_{i-3})$) to get a constant acceleration model. We might reasonably want the new position of the point to depend on $\boldsymbol{p}_{i-4}$ or other points even further back in the history of the point's track; to represent dynamics like this, all we need to do is augment the state vector to a suitable size. Notice that it can be somewhat difficult to visualize how the model will behave. There are two approaches to determining what $\mathcal{D}_i$ needs to be; in the first, we know something about the dynamics and can write it down, as we have done here; in the second, we need to learn it from data — we put discussion of this topic off.

18.2.2 Kalman Filtering

An important feature of the class of models we have described is that all the conditional probability models we need to deal with are normal. In particular, $P(\boldsymbol{X}_i|\boldsymbol{y}_1,\ldots,\boldsymbol{y}_{i-1})$ is normal; as is $P(\boldsymbol{X}_i|\boldsymbol{y}_1,\ldots,\boldsymbol{y}_i)$. This means that they are relatively easy to represent — all we need to do is maintain representations of the mean and the covariance for the prediction and correction phase. In particular, our model will admit a relatively simple process where the representation of the mean and covariance for the prediction and estimation phase are updated.

18.2.3 The Kalman Filter for a 1D State Vector

The dynamic model is now

$$x_i \sim N(d_i x_{i-1}, \sigma_{d_i}^2)$$

$$y_i \sim N(m_i x_i, \sigma_{m_i}^2)$$

We need to maintain a representation of $P(X_i|y_0,\ldots,y_{i-1})$ and of $P(X_i|y_0,\ldots,y_i)$. In each case, we need only represent the mean and the standard deviation, because the distributions are normal.

Notation

We will represent the mean of $P(X_i|y_0,\ldots,y_{i-1})$ as $\overline{X}_i^-$ and the mean of $P(X_i|y_0,\ldots,y_i)$ as $\overline{X}_i^+$ — the superscripts suggest that they represent our belief about X_i immediately before and immediately after the i'th measurement arrives. Similarly, we will represent the standard deviation of $P(X_i|y_0,\ldots,y_{i-1})$ as σ_i^- and of $P(X_i|y_0,\ldots,y_i)$ as σ_i^+. In each case, we will assume that we know $P(X_{i-1}|y_0,\ldots,y_{i-1})$, meaning that we know $\overline{X}_{i-1}^+$ and σ_{i-1}^+.

Tricks with Integrals

The main reason that we work with normal distributions is that their integrals are quite well behaved. We are going to obtain values for various parameters as integrals, usually by change of variable. Our current notation can make appropriate changes a bit difficult to spot, so we write

$$g(x;\mu,v) = \exp\left(-\frac{(x-\mu)^2}{2v}\right)$$

We have dropped the constant, and for convenience are representing the *variance* (as v), rather than the standard deviation. This expression allows some convenient transformations; in particular, we have

$$g(x;\mu,v) = g(x-\mu;0,v)$$

$$
\begin{aligned}
g(m; n, v) &= g(n; m, v) \\
g(ax; \mu, v) &= g(x; \mu/a, v/a^2)
\end{aligned}
$$

We will also need the following fact:

$$\int_{-\infty}^{\infty} g(x-u; \mu, v_a) g(u; 0, v_b) du \propto g(x; \mu, v_a^2 + v_b^2)$$

(there are several ways to confirm that this is true: the easiest is to look it up in tables; more subtle is to think about convolution directly; more subtle still is to think about the sum of two independent random variables). We need a further identity. We have

$$g(x; a, b) g(x; c, d) = g(x; \frac{ad+cb}{b+d}, \frac{bd}{b+d}) f(a, b, c, d)$$

here the form of f is not significant, but the fact that it is not a function of x is. The exercises show you how to prove this identity.

Prediction

We have

$$P(X_i|y_0, \ldots, y_{i-1}) = \int P(X_i|X_{i-1}) P(X_{i-1}|y_0, \ldots, y_{i-1}) dX_{i-1}$$

Now

$$
\begin{aligned}
P(X_i|y_0, \ldots, y_{i-1}) &= \int P(X_i|X_{i-1}) P(X_{i-1}|y_0, \ldots, y_{i-1}) dX_{i-1}) \\
&\propto \int_{-\infty}^{\infty} g(X_i; d_i X_{i-1}, \sigma_{d_i}^2) g(X_{i-1}; \overline{X}_{i-1}^{+}, (\sigma_{i-1}^{+})^2) dX_{i-1} \\
&\propto \int_{-\infty}^{\infty} g((X_i - d_i X_{i-1}); 0, \sigma_{d_i}^2) g((X_{i-1} - \overline{X}_{i-1}^{+}); 0, (\sigma_{i-1}^{+})^2) dX_{i-1} \\
&\propto \int_{-\infty}^{\infty} g((X_i - d_i(u + \overline{X}_{i-1}^{+})); 0, (\sigma_{d_i})^2) g(u; 0, (\sigma_{i-1}^{+})^2) du \\
&\propto \int_{-\infty}^{\infty} g((X_i - d_i u); d_i \overline{X}_{i-1}^{+}, \sigma_{d_i}^2) g(u; 0, (\sigma_{i-1}^{+})^2) du \\
&\propto \int_{-\infty}^{\infty} g((X_i - v); d_i \overline{X}_{i-1}^{+}, \sigma_{d_i}^2) g(v; 0, (d_i \sigma_{i-1}^{+})^2) dv \\
&\propto g(X_i; d_i \overline{X}_0^{+}, \sigma_{d_i}^2 + (d_i \sigma_{i-1}^{+})^2)
\end{aligned}
$$

where we have applied the transformations above, and changed variable twice. All this means that

$$\begin{aligned} \overline{X}_i^- &= d_i \overline{X}_{i-1}^+ \\ (\sigma_i^-)^2 &= \sigma_{d_i}^2 + (d_i \sigma_{i-1}^+)^2 \end{aligned}$$

Correction

We have

$$\begin{aligned} P(X_i|y_0, \ldots, y_i) &= \frac{P(y_i|X_i)P(X_i|y_0, \ldots, y_{i-1})}{\int P(y_i|X_i)P(X_i|y_0, \ldots, y_{i-1})dX_i} \\ &\propto P(y_i|X_i)P(X_i|y_0, \ldots, y_{i-1}) \end{aligned}$$

We know $\overline{X}_i^-$ and σ_i^-, which represent $P(X_i|y_0, \ldots, y_{i-1})$.

Using the notation above, we have

$$\begin{aligned} P(X_i|y_0, \ldots, y_i) &\propto g(y_i; m_i X_i, \sigma_{m_i}^2) g(X_i; \overline{X}_i^-, (\sigma_i^-)^2) \\ &= g(m_i X_i; y_i, \sigma_{m_i}^2) g(X_i; \overline{X}_i^-, (\sigma_i^-)^2) \\ &= g(X_i; \frac{y_i}{m_i}, \frac{\sigma_{m_i}^2}{m_i^2}) g(X_i; \overline{X}_i^-, (\sigma_i^-)^2) \end{aligned}$$

and by pattern matching to the identity above, we have

$$\begin{aligned} X_i^+ &= \left(\frac{\overline{X}_i^- \sigma_{m_i}^2 + m_i y_i (\sigma_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2} \right) \\ \sigma_i^+ &= \sqrt{\left(\frac{\sigma_{m_i}^2 (\sigma_i^-)^2}{(\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2)} \right)} \end{aligned}$$

18.2.4 The Kalman Update Equations for a General State Vector

We obtained a 1D tracker without having to do any integration using special properties of normal distributions. This approach works for a state vector of arbitrary dimension, but the process of guessing integrals, etc., is a good deal more elaborate than that shown in section 18.2.3. We omit the necessary orgy of notation — it's a tough but straightforward exercise for those who really care (you should figure out the identities first and the rest follows) — and simply give the result in algorithm **??**.

Dynamic Model:

$$x_i \sim N(d_i x_{i-1}, \sigma_{d_i})$$

$$y_i \sim N(m_i x_i, \sigma_{m_i})$$

Start Assumptions: $\overline{x}_0^-$ and σ_0^- are known

Update Equations: Prediction

$$\overline{x}_i^- = d_i \overline{x}_{i-1}^+$$

$$\sigma_i^- = \sqrt{\sigma_{d_i}^2 + (d_i \sigma_{i-1}^+)^2}$$

Update Equations: Correction

$$x_i^+ = \left(\frac{\overline{x}_i^- \sigma_{m_i}^2 + m_i y_i (\sigma_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2} \right)$$

$$\sigma_i^+ = \sqrt{\left(\frac{\sigma_{m_i}^2 (\sigma_i^-)^2}{(\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2)} \right)}$$

Algorithm 18.1: *The 1D Kalman filter updates estimates of the mean and covariance of the various distributions encountered while tracking a one-dimensional state variable using the given dynamic model.*

18.2.5 Forward-Backward Smoothing

It is important to notice that $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ is not the best available representation of $\boldsymbol{X}_i$; this is because it doesn't take into account the future behaviour of the point. In particular, all the measurements *after* $\boldsymbol{y}_i$ could affect our representation of $\boldsymbol{X}_i$. This is because these future measurements might contradict the estimates obtained to date — perhaps the future movements of the point are more in agreement with a slightly different estimate of the position of the point. However, $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ *is* the best estimate available at step i.

What we do with this observation depends on the circumstances. If our appli-

Dynamic Model:

$$\begin{aligned} \boldsymbol{x}_i &\sim N(\mathcal{D}_i \boldsymbol{x}_{i-1}, \Sigma_{d_i}) \\ \boldsymbol{y}_i &\sim N(\mathcal{M}_i \boldsymbol{x}_i, \Sigma_{m_i}) \end{aligned}$$

Start Assumptions: $\overline{\boldsymbol{x}}_0^-$ and Σ_0^- are known

Update Equations: Prediction

$$\begin{aligned} \overline{\boldsymbol{x}}_i^- &= \mathcal{D}_i \overline{\boldsymbol{x}}_{i-1}^+ \\ \Sigma_i^- &= \Sigma_{d_i} + \mathcal{D}_i \sigma_{i-1}^+ \mathcal{D}_i \end{aligned}$$

Update Equations: Correction

$$\begin{aligned} \mathcal{K}_i &= \Sigma_i^- \mathcal{M}_i^T \left[\mathcal{M}_i \Sigma_i^- \mathcal{M}_i^T + \Sigma_{m_i} \right]^{-1} \\ \overline{\boldsymbol{x}}_i^+ &= \overline{\boldsymbol{x}}_i^- + \mathcal{K}_i \left[\boldsymbol{y}_i - \mathcal{M}_i \overline{\boldsymbol{x}}_i^- \right] \\ \Sigma_i^+ &= \left[Id - \mathcal{K}_i \mathcal{M}_i \right] \Sigma_i^- \end{aligned}$$

Algorithm 18.2: *The Kalman filter updates estimates of the mean and covariance of the various distributions encountered while tracking a state variable of some fixed dimension using the given dynamic model.*

cation requires an immediate estimate of position — perhaps we are tracking a car in the opposite lane — there isn't much we can do. If we are tracking off-line — perhaps, for forensic purposes, we need the best estimate of what an object was doing given a videotape — then we can use all data points, and so we want to represent $P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$. A common alternative is that we need a rough estimate immediately, and can use an improved estimate that has been time-delayed by a number of steps. This means we want to represent $P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i+k})$ — we have to wait till time $i + k$ for this representation, but it should be an improvement on $P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$.

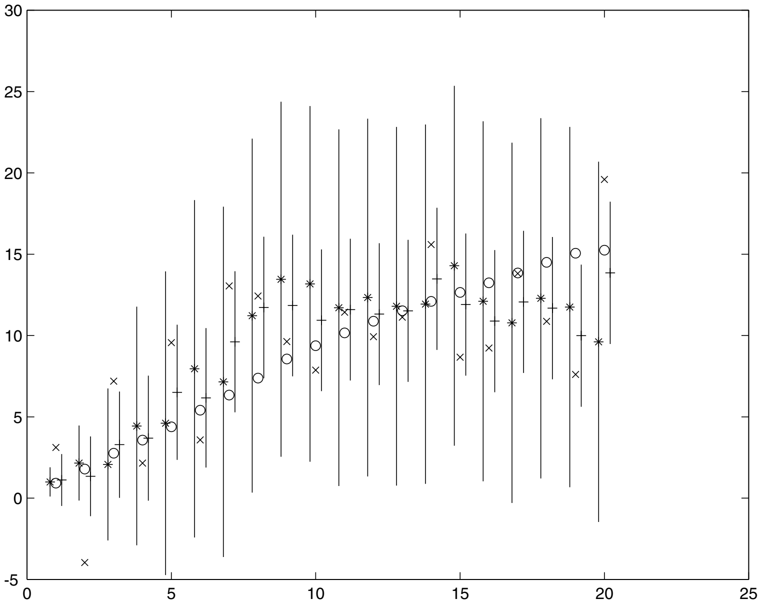


Figure 18.3. The Kalman filter for a point moving on the line under our model of constant velocity (compare with figure 18.1). The state is plotted with open circles, as a function of the step i. The *-s give $\overline{\boldsymbol{x}}_i^-$, which is plotted slightly to the left of the state to indicate that the estimate is made before the measurement. The x-s give the measurements, and the +-s give $\overline{\boldsymbol{x}}_i^+$, which is plotted slightly to the right of the state. The vertical bars around the *-s and the +-s are 3 standard deviation bars, using the estimate of variance obtained before and after the measurement, respectively. When the measurement is noisy, the bars don't contract all that much when a measurement is obtained (compare with figure 18.4).

Introducing a Backward Filter

Now we have

$$\begin{aligned}
P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_N) &= \frac{P(\boldsymbol{X}_i,\boldsymbol{y}_{i+1},\ldots,\boldsymbol{y}_N|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}{P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_N)} \\
&= \frac{P(\boldsymbol{y}_{i+1},\ldots,\boldsymbol{y}_N|\boldsymbol{X}_i,\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}{P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_N)} \\
&= \frac{P(\boldsymbol{y}_{i+1},\ldots,\boldsymbol{y}_N|\boldsymbol{X}_i)P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}{P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_N)} \\
&= P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1},\ldots,\boldsymbol{y}_N)P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)\left(\frac{P(\boldsymbol{y}_{i+1},\ldots,\boldsymbol{y}_N)P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)}{P(\boldsymbol{X}_i)P(\boldsymbol{y}_0,\ldots,\boldsymbol{y}_N)}\right)
\end{aligned}$$

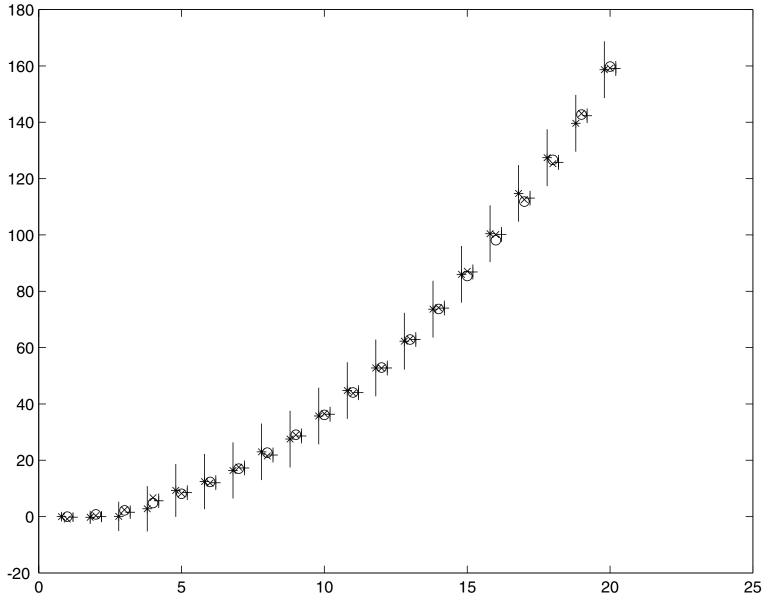


Figure 18.4. The Kalman filter for a point moving on the line under our model of constant acceleration (compare with figure 18.2). The state is plotted with open circles, as a function of the step i. The *-s give $\overline{\boldsymbol{x}}_i^-$, which is plotted slightly to the left of the state to indicate that the estimate is made before the measurement. The x-s give the measurements, and the +-s give $\overline{\boldsymbol{x}}_i^+$, which is plotted slightly to the right of the state. The vertical bars around the *-s and the +-s are 3 standard deviation bars, using the estimate of variance obtained before and after the measurement, respectively. When the measurement is noisy, the bars don't contract all that much when a measurement is obtained.

The fraction in brackets should look like a potential source of problems to you; in fact, we will be able to avoid tangling with it by a clever trick. What is important about this form is that we are combining $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ — which we know how to obtain — with $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_N)$. We actually know how to obtain a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_N)$, too. We could simply run the Kalman filter *backwards* in time, using *backward dynamics*, and take the predicted representation of $\boldsymbol{X}_i$ (we leave the details of relabelling the sequence, etc. to the exercises).

Combining Representations

Now we have two representations of $\boldsymbol{X}_i$: one obtained by running a forward filter, and incorporating all measurements up to $\boldsymbol{y}_i$; and one obtained by running a backward filter, and incorporating all measurements after $\boldsymbol{y}_i$. We need to combine these representations. Instead of explicitly determining the missing terms in equation **??**, we can get the answer by noting that *this is like having another measurement.* In particular, we have a new measurement generated by $\boldsymbol{X}_i$ — that is, the result of the backward filter — to combine with our estimate from the forward filter. We know how to combine estimates with measurements, because that's what the Kalman filter equations are for.

All we need is a little notation. We will attach the superscript f to the estimate from the forward filter, and the superscript b to the estimate from the backward filter. We will write the mean of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$ as $\overline{\boldsymbol{X}}_i^*$ and the covariance of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$ as Σ_i^*. We regard the representation of $\boldsymbol{X}_i^b$ as a measurement of $\boldsymbol{X}_i$ with mean $\overline{\boldsymbol{X}}_i^{b,-}$ and covariance $\Sigma_i^{b,-}$ — the minus sign is because the i'th measurement cannot be used twice, meaning the backward filter predicts $\boldsymbol{X}_i$ using $\boldsymbol{y}_N \ldots \boldsymbol{y}_{i+1}$. This measurement needs to be combined with $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$, which has mean $\overline{\boldsymbol{X}}_i^{f,+}$ and covariance $\Sigma_i^{f,+}$ (when we substitute into the Kalman equations, these will take the role of the representation *before* a measurement, because the value of the measurement is now $\overline{\boldsymbol{X}}_i^{b,-}$).

Substituting into the Kalman equations, we find

$$\begin{aligned}
\mathcal{K}_i^* &= \Sigma_i^{f,+}\left[\Sigma_i^{f,+} + \Sigma_i^{b,-}\right]^{-1} \\
\Sigma_i^* &= \left[I - \mathcal{K}_i\right]\Sigma_i^{+,f} \\
\overline{\boldsymbol{X}}_i^* &= \overline{\boldsymbol{X}}_i^{f,+} + \mathcal{K}_i^*\left[\overline{\boldsymbol{X}}_i^{b,-} - \overline{\boldsymbol{X}}_i^{f,+}\right]
\end{aligned}$$

It turns out that a little manipulation (exercises!) yields a simpler form, which we give in algorithm 3. Forward-backward estimates can make a substantial difference, as figure 18.5 illustrates.

Priors

In typical vision applications, we are tracking forward in time. This leads to an inconvenient asymmetry: we may have a good idea of where the object started, but only a poor one of where it stopped, i.e. we are likely to have a fair prior for $P(\boldsymbol{x}_0)$, but may have difficulty supplying a prior for $P(\boldsymbol{x}_N)$ for the forward-backward filter. One option is to use $P(\boldsymbol{x}_N|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_N)$ as a prior. This is a dubious act, as this probability distribution does not in fact reflect our prior belief about $P(\boldsymbol{x}_N)$ — we've used all the measurements to obtain it. The consequences can be that this distribution understates our uncertainty in $\boldsymbol{x}_N$, and so leads to a forward-backward estimate that significantly underestimates the covariance for the later states. An alternative is to use a the mean supplied by the forward filter, but enlarge the covariance substantially; the consequences are a forward-backward estimate that overestimates the covariance for the later states (compare figure 18.5 with figure 18.6).

Not all applications have this asymmetry; for example, if we are engaged in a forensic study of a videotape, we might be able to start both the forward tracker

Forward filter:
Obtain the mean and variance of $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ using the Kalman filter. These are $\overline{\boldsymbol{X}}_i^{f,+}$ and $\Sigma_i^{f,+}$.

Backward filter: Obtain the mean and variance of $P(\boldsymbol{X}_i|\boldsymbol{y}_{i+1}, \ldots, \boldsymbol{y}_N)$ using the Kalman filter running backwards in time. These are $\overline{\boldsymbol{X}}_i^{b,-}$ and $\Sigma_i^{b,-}$.

Combining forward and backward estimates: Regard the backward estimate as a new measurement for $\boldsymbol{X}_i$, and insert into the Kalman filter equations to obtain

$$\begin{aligned}\Sigma_i^* &= \left[(\Sigma_i^{f,+})^{-1} + (\Sigma_i^{b,-})^{-1}\right]^{-1} \\ \overline{\boldsymbol{X}}_i^* &= \Sigma_i^* \left[(\Sigma_i^{f,+})^{-1}\overline{\boldsymbol{X}}_i^{f,+} + (\Sigma_i^{b,-})^{-1}\overline{\boldsymbol{X}}_i^{b,-}\right]\end{aligned}$$

Algorithm 18.3: *The forward backward algorithm combines forward and backward estimates of state to come up with an improved estimate.*

and the backward tracker by hand, and provide a good estimate of the prior in each case. If this is possible, then we have a good deal more information which may be able to help choose correspondences, etc. — the forward tracker should finish rather close to where the backward tracker starts.

Smoothing over an Interval

While our formulation of forward-backward smoothing assumed that the backward filter started at the last data point, it is easy to start this filter a fixed number of steps ahead of the forward filter. If we do this, we obtain an estimate of state in real time (essentially immediately after the measurement), and an improved estimate some fixed numbers of measurements later. This is sometimes useful. Furthermore, it is an efficient way to obtain most of the improvement available from a backward filter, if we can assume that the effect of the distant future on our estimate is relatively small compared with the effect of the immediate future. Notice that we need to be careful about priors for the backward filter here; we might take the forward estimate and enlarge its covariance somewhat.

18.3 Non-Linear Dynamic Models

If we can assume that noise is normally distributed, linear dynamic models are reasonably easy to deal with, because a linear map takes a random variable with a normal distribution to another random variable with a (different, but easily deter-

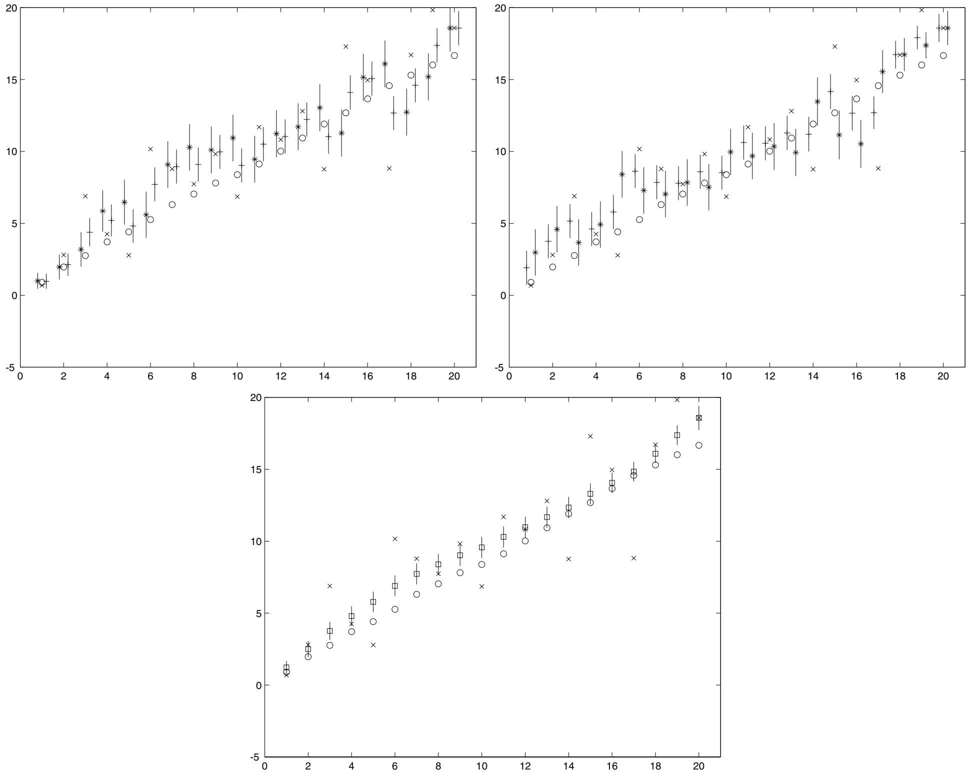


Figure 18.5. Forward-backward estimation for a dynamic model of a point moving on the line with constant velocity. We are plotting the position component of state against time. On the **top left**, we show the forward estimates, again using the convention that the state is shown with circles, the data is shown with x's, the prediction is shown with a * and the corrected estimate is shown with a +; the bars give one standard deviation in the estimate. The predicted estimate is shown slightly behind the state and the corrected estimate is shown slightly ahead of the state. You should notice that the measurements are very noisy. On the **top right** we show the backward estimates. Now time is running backwards (although we have plotted both curves on the same axis) so that the prediction is slightly ahead of the measurement and the corrected estimate is slightly behind. We have used the final corrected estimate of the forward filter as a prior; again, the bars give one standard deviation in each variable. On the **bottom**, we show the combined forward-backward estimate. The squares give the estimates of state. Notice the significant improvement in the estimate.

mined) normal distribution. We used this fact extensively in describing the Kalman filter. Because we knew that everything was normal, we could do most calculations by determining the mean and covariance of the relevant normal distribution, a process that is often quite easy if one doesn't try to do the integrals directly. Fur-

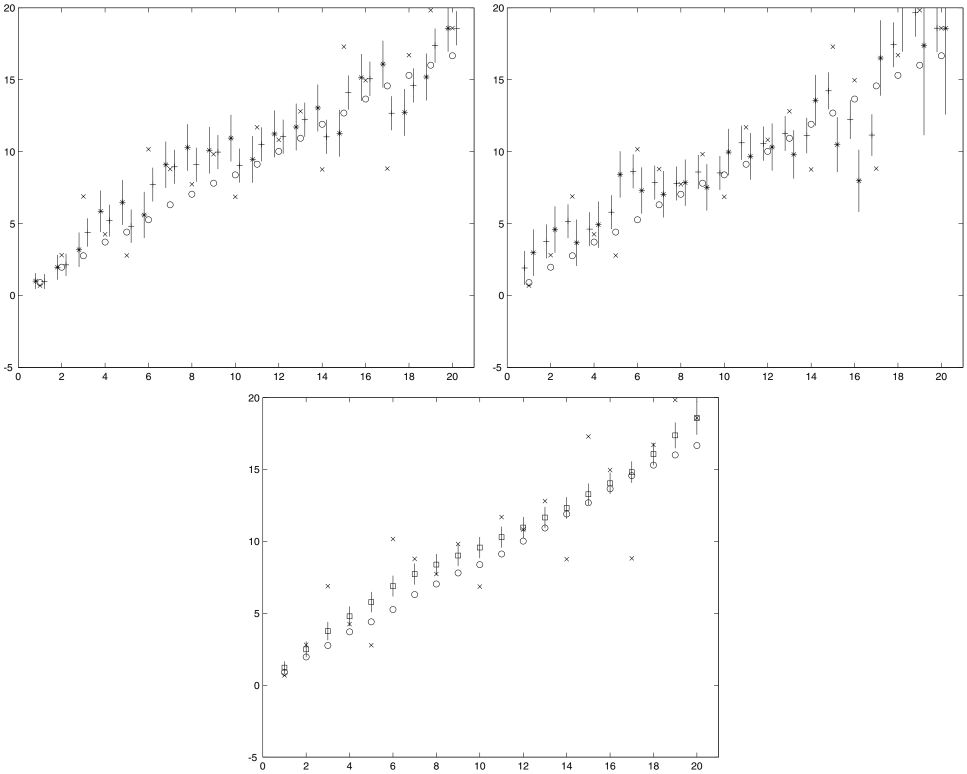


Figure 18.6. We now show the effects of using a diffuse prior for the position of the final point in forward-backward estimation for a dynamic model of a point moving on the line with constant velocity. We are plotting the position component of state against time. On the **top left**, we show the forward estimates, again using the convention that the state is shown with circles, the data is shown with x's, the prediction is shown with a * and the corrected estimate is shown with a +; the bars give one standard deviation in the estimate. The predicted estimate is shown slightly behind the state and the corrected estimate is shown slightly ahead of the state. You should notice that the measurements are very noisy. On the **top right** we show the backward estimates. Now time is running backwards (although we have plotted both curves on the same axis) so that the prediction is slightly ahead of the measurement and the corrected estimate is slightly behind; again, the bars give one standard deviation in each variable. On the **bottom**, we show the combined forward-backward estimate. The squares give the estimates of state. Notice the significant improvement in the estimate.

thermore, because a normal distribution is represented by its mean and covariance, we knew what representation of the relevant distributions to maintain.

Many natural dynamic models are non-linear. There are two sources of problems.

Firstly, in models where the dynamics have the form

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i); \Sigma_{d_i})$$

(where $\boldsymbol{f}$ is a non-linear function), both $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ and $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ tend not to be normal. As section 18.3.1 will show, even quite innocuous looking nonlinearities can lead to very strange distributions indeed. Secondly, $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ may not be Gaussian either. This phenomenon is quite common in vision; it leads to difficulties that are still neither well understood nor easily dealt with (section 18.3.2).

Dealing with these phenomena is difficult. There is not, and will never be, a completely general solution. It is always a good idea to see if a linear model can be made to work. If one does not, there is the option of linearizing the model locally and assuming that everything is normal. This approach, known as the **extended Kalman filter** tends to be unreliable in many applications. We describe it briefly in the appendix, because it is useful on occasion. Finally, there is a method that maintains a radically different representation of the relevant distributions from that used by the Kalman filter. This method is described in section **??**. The rest of this section illustrates some of the difficulties presented by non-linear problems.

18.3.1 Unpleasant Properties of Non-Linear Dynamics

Non-linear models of state evolution can take unimodal distributions — like Gaussians — and create multiple, well-separated modes, phenomena that are very poorly modeled by a single Gaussian.

This effect is most easily understood by looking at an example. Let us have the (apparently simple) dynamical model $x_{i+1} = x_i + 0.1 * \sin x_i$. Notice that there is no random component to this dynamical model at all; now let us consider $P(X_1)$, assuming that $P(X_0)$ is a Gaussian with very large variance (and so basically flat over a large range). The easiest way to think about this problem is to consider what happens to various points; as figure 18.7 illustrates, points in the range $((2k)\pi, (2k+2)\pi)$ move towards $(2k+1)\pi$. This means that probability must collect at points $(2k+1)\pi$ (we ask you to provide some details in the exercises).

This nonlinearity is apparently very small. Its effects are very substantial, however. One way to see what happens is to follow a large number of different points through the dynamics for many steps. We choose a large collection of points according to $P(X_0)$, and then apply our dynamic model to them. A histogram of these points at each step provides a rough estimate of $P(X_i)$, and we can plot how they evolve, too; the result is illustrated in figure 18.8. As this figure shows, $P(X_i)$ very quickly looks like a set of narrow peaks, each with a different weight, at $(2k+1)\pi$. Representing this distribution by reporting only its mean and covariance involves a substantial degree of wishful thinking.

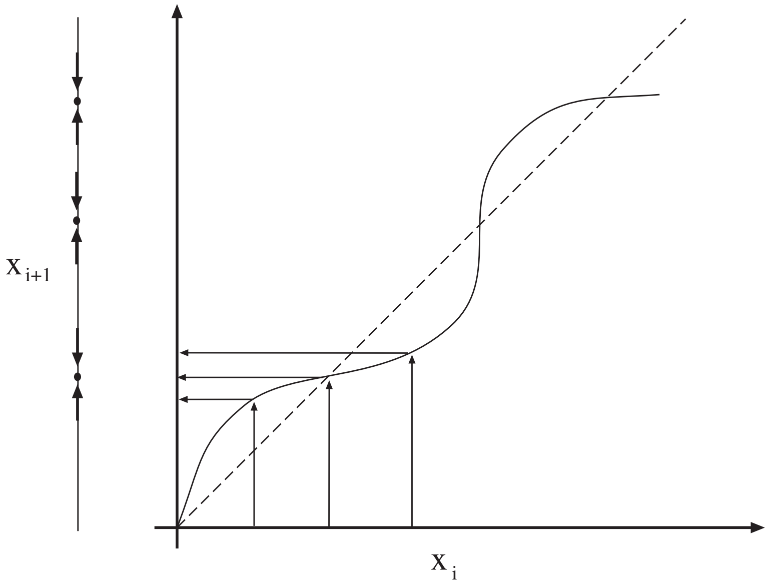


Figure 18.7. The non-linear dynamics $x_{i+1} = x_i + 0.1 \sin x_i$ cause points n the range $((2k)\pi, (2k+2)\pi)$ move towards $(2k+1)\pi$. As the figure on the **left** illustrates, this is because $x_i + 0.1 \sin x_i$ is slightly smaller than x_i for x_i in the range $((2k+1)\pi, (2k+2)\pi)$ and is slightly larger than x_i for x_i in the range $((2k)\pi, (2k+1)\pi)$. In fact, the nonlinearity of this function looks small — it is hardly visible in a scaled plot. However, as figure **??** shows, its effects are very significant.

18.3.2 Difficulties with Likelihoods

There is another reason to believe that $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ may be very complicated in form. Even if the dynamics do not display the effects of section 18.3.1, the likelihood function $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ can create serious problems. For many important cases we expect that the likelihood has multiple peaks. For example, consider tracking people in video sequences. The state will predict the configuration of an idealised human figure and $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ will be computed by comparing predictions about the image with the actual image, in some way. As the configuration of the idealised human figure changes, it will cover sections of image that aren't generated by a person but look as though they are. For example, pretty much any coherent long straight image region with parallel sides can look like a limb — this means that as $\boldsymbol{X}$ changes to move the arm of the idealised figure from where it should be to cover this region, the value of $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ will go down, and then up again. The likely result is a function $P(\boldsymbol{Y}_i|\boldsymbol{X}_i)$ with many peaks in it.

We will almost certainly need to keep track of more than one of these peaks This is because the largest peak for any given frame may not always correspond to the *right* peak. This ambiguity should resolve itself once we have seen some

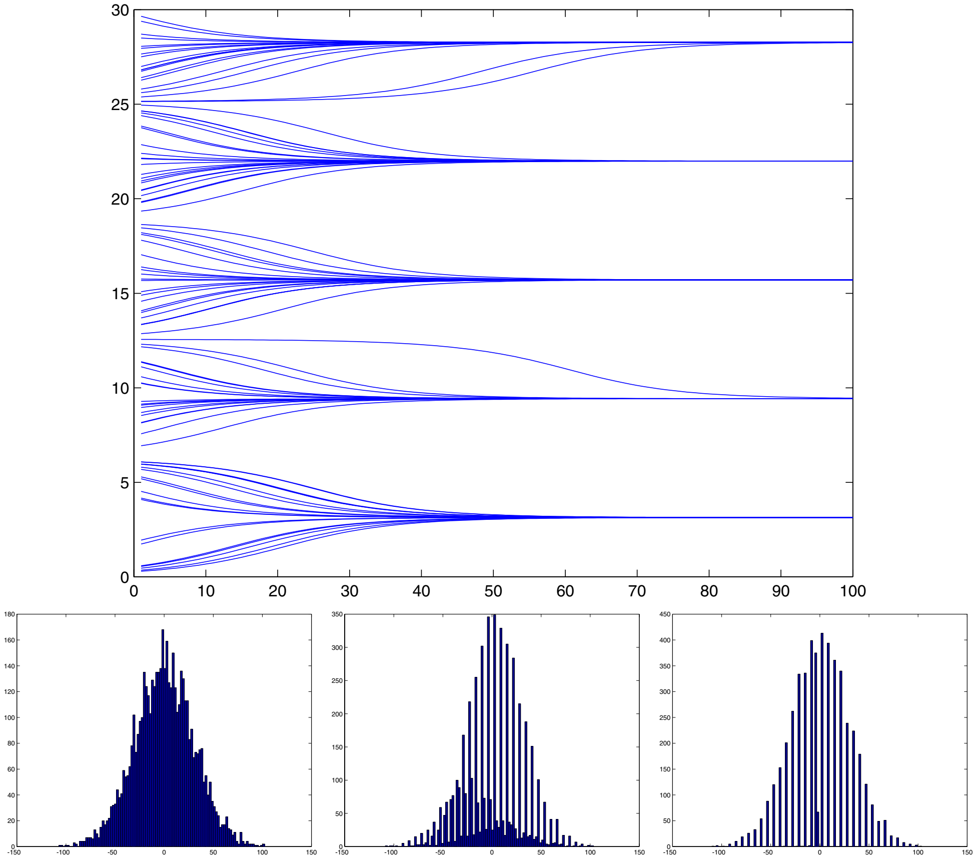


Figure 18.8. On the **top**, we have plotted the time evolution of the state of a set of 100 points, for 100 steps of the process $x_{i+1} = x_i + 0.1 * \sin x_i$. Notice that the points all contract rather quickly to $(2k+1)\pi$, and stay there. We have joined up the tracks of the points to make it clear how the state changes. On the **bottom left** we show a histogram of the start states of the points we used; this is an approximation to $P(x_0)$. The histogram on the **bottom center** shows a histogram of the point positions after 20 iterations; this is an approximation to $P(x_{20})$. The histogram on the **bottom right** shows a histogram of the point positions after 70 iterations; this is an approximation to $P(x_{70})$. Notice that there are many important peaks to this histogram — it might be very unwise to model $P(x_i)$ as a Gaussian.

more frames — we don't expect to see many image assemblies that look like people, move like people for many frames and yet aren't actually people. However, until it does, we may need to manage a representation of $P(\boldsymbol{X}_i|\boldsymbol{y}_0,\ldots,\boldsymbol{y}_i)$ which contains several different peaks. This presents considerable algorithmic difficulties — we don't know how many peaks there are, or where they are, and finding them in a high dimensional space may be difficult. One partially successful approach is a form of random search, known as **particle filtering**.

18.4 Particle Filtering

The main difficulty in tracking in the presence of complicated likelihood functions or of non-linear dynamics is in maintaining a satisfactory representation of $P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$. This representation should be handle multiple peaks in the distribution, and should be able to handle a high-dimensional state vector without difficulty. There is no completely satisfactory general solution to this problem (and there will never be). In this section, we discuss an approach that has been useful in many applications.

18.4.1 Sampled Representations of Probability Distributions

A natural way to think about representations of probability distributions is to ask what a probability distribution is for. Computing a representation of a probability distributions is not our primary objective; we wish to represent a probability distribution so that we can compute one or another expectation. For example, we might wish to compute the expected state of an object given some information; we might wish to compute the variance in the state, or the expected utility of shooting at an object, etc. Probability distributions are devices for computing expectations — thus, our representation should be one that gives us a decent prospect of computing an expectation accurately. This means that there is a strong resonance between questions of representing probability distributions and questions of efficient numerical integration.

Monte Carlo Integration using Importance Sampling

Assume that we have a collection of N points $\boldsymbol{u}^i$, and a collection of weights w^i. These points are independent samples drawn from a probability distribution $S(\boldsymbol{U})$ — we call this the **sampling distribution**; notice that we have broken with our usual convention of writing any probability distribution with a P. We assume that $S(\boldsymbol{U})$ has a probability density function $s(\boldsymbol{U})$.

The weights have the form $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$ for some function f. Now it is a fact that

$$\begin{aligned} \mathrm{E}\left[\frac{1}{N}\sum_i g(\boldsymbol{u}^i)w^i\right] &= \int g(\boldsymbol{U})\frac{f(\boldsymbol{U})}{s(\boldsymbol{U})}s(\boldsymbol{U})d\boldsymbol{U} \\ &= \int g(\boldsymbol{U})f(\boldsymbol{U})d\boldsymbol{U} \end{aligned}$$

where the expectation is taken over the distribution on the collection of N independent samples from $S(\boldsymbol{U})$ (you can prove this fact using the weak law of large numbers []).

`technical assumptions go here`

The variance of this estimate goes down as $1/N$, and is independent of the dimension of $\boldsymbol{U}$.

Representing Distributions using Weighted Samples

If we think about a distribution as a device for computing expectations — which are integrals — we can obtain a representation of a distribution from the integration method described above. This representation will consist of a set of weighted points. Assume that f is non-negative, and $\int f(\boldsymbol{U}) d\boldsymbol{U}$ exists and is finite. Then

$$\frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U}) d\boldsymbol{U}}$$

is a probability density function representing the distribution of interest. We shall write this probability density function as $p_f(\boldsymbol{X})$.

Now we have a collection of N points $\boldsymbol{u}^i \sim S(\boldsymbol{U})$, and a collection of weights $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Using this notation, we have that

$$\begin{aligned} \mathrm{E}\left[\frac{1}{N}\sum_i w^i\right] &= \int 1 \frac{f(\boldsymbol{U})}{s(\boldsymbol{U})} s(\boldsymbol{U}) d\boldsymbol{U} \\ &= \int f(\boldsymbol{U}) d\boldsymbol{U} \end{aligned}$$

Now this means that

$$\begin{aligned} \mathrm{E}_{p_f}[g] &= \int g(\boldsymbol{U}) p_f(\boldsymbol{U}) d\boldsymbol{U} \\ &= \frac{\int g(\boldsymbol{U}) f(\boldsymbol{U}) d\boldsymbol{U}}{\int f(\boldsymbol{U}) d\boldsymbol{U}} \\ &= \mathrm{E}\left[\frac{\sum_i g(\boldsymbol{u}_i) w_i}{\sum_i w_i}\right] \\ &\approx \frac{\sum_i g(\boldsymbol{u}_i) w_i}{\sum_i w_i} \end{aligned}$$

(where we have cancelled some N's). This means that we can *in principle* represent a probability distribution by a set of weighted samples (algorithm 4). There are some significant practical issues here, however. Before we explore these, we will discuss how to perform various computations with sampled representations. We have already shown how to compute an expectation (above, and algorithm 5). There are two other important activities for tracking: marginalisation, and turning a representation of a prior into a representation of a posterior.

Represent a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of N weighted samples

$$\{(\boldsymbol{u}^i, w^i)\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$.

Algorithm 18.4: *Obtaining a sampled representation of a probability distribution*

We have a representation of a probability distribution

$$p_f(\boldsymbol{X}) = \frac{f(\boldsymbol{X})}{\int f(\boldsymbol{U})d\boldsymbol{U}}$$

by a set of weighted samples

$$\{(\boldsymbol{u}^i, w^i)\}$$

where $\boldsymbol{u}^i \sim s(\boldsymbol{u})$ and $w^i = f(\boldsymbol{u}^i)/s(\boldsymbol{u}^i)$. Then:

$$\int g(\boldsymbol{U})p_f(\boldsymbol{U})d\boldsymbol{U} \approx \frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i)w^i}{\sum_{i=1}^{N} w^i}$$

Algorithm 18.5: *Computing an expectation using a set of samples*

Marginalising a Sampled Representation

An attraction of sampled representations is that some computations are particularly easy. Marginalisation is a good and useful example. Assume we have a sampled representation of $p_f(\boldsymbol{U}) = p_f((\boldsymbol{M}, \boldsymbol{M}))$. We write $\boldsymbol{U}$ as two components $(\boldsymbol{M}, \boldsymbol{N})$ so that we can marginalise with respect to one of them.

Now assume that the sampled representation consists of a set of samples which we can write as

$$\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\}$$

In this representation, $(\boldsymbol{m}^i, \boldsymbol{n}^i) \sim s(\boldsymbol{M}, \boldsymbol{N})$ and $w^i = f((\boldsymbol{m}^i, \boldsymbol{n}^i))/s((\boldsymbol{m}^i, \boldsymbol{n}^i))$.

We want a representation of the marginal $p_f(\boldsymbol{M}) = \int p_f(\boldsymbol{M}, \boldsymbol{N})d\boldsymbol{N}$. We will use this marginal to estimate integrals, so we can derive the representation by thinking about integrals. In particular

$$\int g(\boldsymbol{M})p_f(\boldsymbol{M})d\boldsymbol{M} = \int g(\boldsymbol{M}) \int p_f(\boldsymbol{M}, \boldsymbol{N})d\boldsymbol{N}d\boldsymbol{M}$$

$$
\begin{aligned}
&= \int \int g(\boldsymbol{M}) p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N} d\boldsymbol{M} \\
&\approx \frac{\sum_{i=1}^{N} g(\boldsymbol{m}^i) w^i}{\sum_{i=1}^{N} w^i}
\end{aligned}
$$

meaning that we can represent the marginal by dropping the $\boldsymbol{n}^i$ components of the sample (or ignoring them, which may be more efficient!).

Assume we have a sampled representation of a distribution

$$p_f(\boldsymbol{M}, \boldsymbol{N})$$

given by

$$\left\{((\boldsymbol{m}^i, \boldsymbol{n}^i), w^i)\right\}$$

Then

$$\left\{(\boldsymbol{m}^i, w^i)\right\}$$

is a representation of the marginal,

$$\int p_f(\boldsymbol{M}, \boldsymbol{N}) d\boldsymbol{N}$$

Algorithm 18.6: *Computing a representation of a marginal distribution*

Transforming a Sampled Representation of a Prior into a Sampled Representation of a Posterior

Appropriate manipulation of the weights of a sampled distribution yields representations of other distributions. A particularly interesting case is representing a posterior, given some measurement. Recall that

$$
\begin{aligned}
p(\boldsymbol{U} | \boldsymbol{V} = v_0) &= \frac{p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{U}) p(\boldsymbol{U})}{\int p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{U}) p(\boldsymbol{U}) d\boldsymbol{U}} \\
&= \frac{1}{K} p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{U}) p(\boldsymbol{U})
\end{aligned}
$$

where $\boldsymbol{v}_0$ is some measured value taken by the random variable $\boldsymbol{V}$.

Assume we have a sampled representation of $p(\boldsymbol{U})$, given by $\left\{(\boldsymbol{u}^i, w^i)\right\}$. We can evaluate K fairly easily:

$$
\begin{aligned}
K &= \int p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{U}) p(\boldsymbol{U}) d\boldsymbol{U} \\
&= \mathrm{E}\left[\frac{\sum_{i=1}^{N} p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i}{\sum_{i=1}^{N} w^i}\right]
\end{aligned}
$$

$$\approx \frac{\sum_{i=1}^{N} p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i}{\sum_{i=1}^{N} w^i}$$

Now let us consider the posterior.

$$\begin{aligned}\int g(\boldsymbol{U}) p(\boldsymbol{U} | \boldsymbol{V} = v_0) d\boldsymbol{U} &= \frac{1}{K} \int g(\boldsymbol{U}) p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{U}) p(\boldsymbol{U}) d\boldsymbol{U} \\ &\approx \frac{1}{K} \frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i) p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i}{\sum_{i=1}^{N} w^i} \\ &\approx \frac{\sum_{i=1}^{N} g(\boldsymbol{u}^i) p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i}{\sum_{i=1}^{N} p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i}\end{aligned}$$

(where we substituted the approximate expression for K in the last step). This means that, if we take $\{(\boldsymbol{u}^i, w^i)\}$ and replace the weights with

$$w'^i = p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i$$

the result $\{(\boldsymbol{u}^i, w'^i)\}$ is a representation of the posterior.

Assume we have a representation of $p(\boldsymbol{U})$ as

$$\{(\boldsymbol{u}^i, w^i)\}$$

Assume we have an observation $\boldsymbol{V} = \boldsymbol{v}_0$, and a likelihood model $p(\boldsymbol{V} | \boldsymbol{U})$. The posterior, $p(\boldsymbol{U} | \boldsymbol{V} = \boldsymbol{v}_0)$ is represented by

$$\{(\boldsymbol{u}^i, w'^i)\}$$

where

$$w'^i = p(\boldsymbol{V} = \boldsymbol{v}_0 | \boldsymbol{u}^i) w^i$$

Algorithm 18.7: *Transforming a sampled representation of a prior into a sampled representation of a posterior.*

18.4.2 The Simplest Particle Filter

Assume that we have a sampled representation of $P(\boldsymbol{X}_{i-1} | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$, and we need to obtain a representation of $P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$. We will follow the usual two steps of prediction and correction.

We can regard each sample as a possible state for the process at step $\boldsymbol{X}_{i-1}$. We are going to obtain our representation by firstly representing

$$P(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

and then marginalising out $\boldsymbol{X}_{i-1}$ (which we know how to do). The result is the prior for the next state, and, since we know how to get posteriors from priors, we will obtain $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$.

Prediction

Now

$$p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) = p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

Write our representation of $p(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\left\{(\boldsymbol{u}^k_{i-1}, w^k_{i-1})\right\}$$

(the superscripts index the samples for a given step i, and the subscript gives the step).

Now for any given sample $\boldsymbol{u}^k_{i-1}$, we can obtain samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ fairly easily. This is because our dynamic model is

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{x}_{i-1}) + \xi_i$$

where $\xi_i \sim N(0, \Sigma_{m_i})$. Thus, for any given sample $\boldsymbol{u}^k_{i-1}$, we can generate samples of $p(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{u}^k_{i-1})$ as

$$\left\{(f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, 1)\right\}$$

where $\xi^l_i \sim N(0, \Sigma_{m_i})$. The index l indicates that we might generate several such samples for each $\boldsymbol{u}^k_{i-1}$.

We can now represent $p(\boldsymbol{X}_i, \boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as

$$\left\{((f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, \boldsymbol{u}^k_{i-1}), w^k_{i-1})\right\}$$

(notice that there are *two* free indexes here, k and l; by this we mean that, for each sample indexed by k, there might be several different elements of the set, indexed by l).

Because we can marginalise by dropping elements, the representation of $P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ is given by

$$\left\{(f(\boldsymbol{u}^k_{i-1}) + \xi^l_i, w^k_{i-1})\right\}$$

(we walk through a proof in the exercises). We will reindex this collection of samples — which may have more than N elements — and rewrite it as

$$\left\{(\boldsymbol{u}^{k,-}_i, w^{k,-}_i)\right\}$$

assuming that there are M elements. Just as in our discussion of Kalman filters, the superscript '$-$' indicates that this our representation of the i'th state before a measurement has arrived. The superscript k gives the individual sample.

Correction

Correction is simple: we need to take the prediction, which acts as a prior, and turn it into a posterior. We do this by choosing an appropriate weight for each sample, following algorithm **??**. The weight is

$$p(\boldsymbol{Y}_i = \boldsymbol{y}_i | \boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}$$

(you should confirm this by comparing with algorithm **??**). and our representation of the posterior is

$$\left\{ (\boldsymbol{s}_i^{k,-}, p(\boldsymbol{Y}_i = \boldsymbol{y}_i | \boldsymbol{X}_i = \boldsymbol{s}_i^{k,-}) w_i^{k,-}) \right\}$$

The Tracking Algorithm

In principle, we now have most of a tracking algorithm — the only missing step is to explain where the samples of $p(\boldsymbol{X}_0)$ came from. The easiest thing to do here is to start with a diffuse prior of a special form that is easily sampled — a Gaussian with large covariance might do it — and give each of these samples a weight of 1. It is a good idea to implement this tracking algorithm to see how it works (exercises!); you will notice that it works poorly even on the simplest problems (figure 18.9 compares estimates from this algorithm to exact expectations computed with a Kalman filter). The algorithm gives bad estimates because most samples represent no more than wasted computation. In jargon, the samples are called **particles**.

If you implement this algorithm, you will notice that weights get small very fast; this isn't obviously a problem, because the mean value of the weights is cancelled in the division, so we could at each step divide the weights by their mean value. If you implement this step, you will notice that very quickly one weight becomes close to one and all others are extremely small. It is a fact that, in the simple particle filter, the variance of the weights cannot decrease with i (meaning that, in general, it will increase and we will end up with one weight very much larger than all others).

If the weights are small, our estimates of integrals are likely to be poor. In particular, a sample with a small weight is positioned at a point where $f(\boldsymbol{u})$ is much smaller than $p(\boldsymbol{u})$; in turn (unless we want to take an expectation of a function which is very large at this point) this sample is likely to contribute relatively little to the estimate of the integral.

Generally, the way to get accurate estimates of integrals is to have samples that lie where the integral is likely to be large — we certainly don't want to miss these points. We are unlikely to want to take expectations of functions that vary quickly, and so we would like our samples to lie where $f(\boldsymbol{u})$ is large. In turn, this means that a sample whose weight w is small represents a waste of resources — we'd rather replace it with another sample with a large weight. This means that the effective number of samples is decreasing — some samples make no significant contribution to the expectations we might compute, and should ideally be replaced (figure 18.9 illustrates this important effect). In the following section, we describe ways of maintaining the set of particles that lead to effective and useful particle filters.

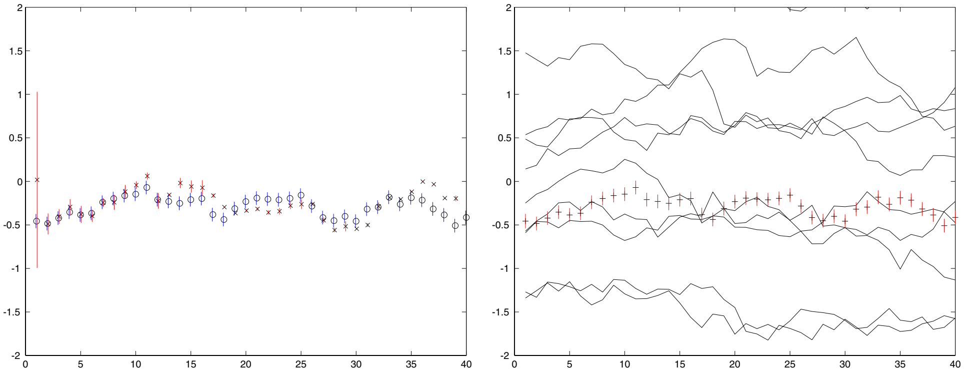


Figure 18.9. The simple particle filter behaves very poorly, as a result of a phenomenon called **sample impoverishment**, which is rather like quantisation error. In this example, we have a point on the line drifting on the line (i.e. $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise. In this case, we can get an exact representation of the posterior using a Kalman filter. In the figure on the **left**, we compare a representation obtained exactly using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a *one* standard deviation bar (previously we used three standard deviations, but that would make these figures difficult to interpret). The mean obtained using a Kalman filter is given as an x; the mean obtained using a particle filter is given as an o; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that the mean is poor, but the standard deviation estimate is awful, and gets worse as the tracking proceeds. In particular, the standard deviation estimate woefully underestimates the standard deviation — this could mislead a user into thinking the tracker was working and producing good estimates, when in fact it is hopelessly confused. The figure on the **right** indicates what is going wrong; we plot the tracks of ten particles, randomly selected from the 100 used. Note that relatively few particles ever lie within one standard deviation of the mean of the posterior; in turn, this means that our representation of $P(x_{i+1}|y_0, \ldots, y_0)$ will tend to consist of many particles with very low weight, and only one with a high weight. This means that the density is represented very poorly, and the error propagates.

18.4.3 A Workable Particle Filter

Particles with very low weights are fairly easily dealt with — we will adjust the collection of particles to emphasize those that appear to be most helpful in representing the posterior. This will help us deal with another difficulty, too. In discussing the simple particle filter, we did not discuss how many samples there were at each stage — if, at the prediction stage, we drew several samples of $P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{s}_{i-1}^{k,+})$ for each $\boldsymbol{s}_{i-1}^{k,+}$, the total pool of samples would grow as i got bigger. Ideally, we would have a constant number of particles N. All this suggests that we need a method to discard samples, ideally concentrating on discarding unhelpful samples. There are a number of strategies that are popular.

Resampling the Prior

At each step i, we have a representation of $P(\boldsymbol{X}_{i-1}|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ via weighted samples. This representation consists of N (possibly distinct) samples, each with an associated weight. Now in a sampled representation, the frequency with which samples appear can be traded off against the weight with which they appear. For example, assume we have a sampled representation of $P(\boldsymbol{U})$ consisting of N pairs $(\boldsymbol{s}_k, w_k)$. Form a new set of samples consisting of a union of N_k copies of $(\boldsymbol{s}_k, 1)$, for each k. If

$$\frac{N_k}{\sum_k N_k} = w_k$$

this new set of samples is also a representation of $P(\boldsymbol{U})$ (you should check this).

Furthermore, if we take a sampled representation of $P(\boldsymbol{U})$ using N samples, and draw N' elements from this set with replacement, uniformly and at random, the result will be a representation of $P(\boldsymbol{U})$, too (you should check this, too). This suggests that we could (a) expand the sample set and then (b) subsample it to get a new representation of $P(\boldsymbol{U})$. This representation will tend to contain multiple copies of samples that appeared with high weights in the original representation.

This procedure is equivalent to the rather simpler process of making N draws with replacement from the original set of samples, using the weights w_i as the probability of drawing a sample. Each sample in the new set would have weight 1; the new set would predominantly contain samples that appeared in the old set with large weights. This process of resampling might occur at every frame, or only when the variance of the weights is too high.

Resampling Predictions

A slightly different procedure is to generate several samples of $P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1} = \boldsymbol{s}^{k,+}_{i-1})$ for each $\boldsymbol{s}^{k,+}_{i-1}$, make N draws, with replacement, from this set using the weights w_i as the probability of drawing a sample, to get N particles. Again, this process will emphasize particles with larger weight over those with smaller weights.

The Consequences of Resampling

Figure 18.10 illustrates the improvements that can be obtained by resampling. Resampling is not a uniformly benign activity, however: it is possible — but unlikely — to lose important particles as a result of resampling, and resampling can be expensive computationally if there are many particles.

18.4.4 If's, And's and But's — Practical Issues in Building Particle Filters

Particle filters have been extremely successful in many practical applications in vision, but can produce some nasty surprises. One important issue has to do with the number of particles; while the expected value of an integral estimated with

Initialization: Represent $P(\boldsymbol{X}_0)$ by a set of N samples

$$\left\{(\boldsymbol{s}_0^{k,-}, w_0^{k,-})\right\}$$

where

$$\boldsymbol{s}_0^{k,-} \sim P_s(\boldsymbol{S})$$

and

$$w_0^{k,-} = P(\boldsymbol{s}_0^{k,-})/P_s(\boldsymbol{S} = \boldsymbol{s}_0^{k,-})$$

Ideally, $P(\boldsymbol{X}_0)$ has a simple form and $\boldsymbol{s}_0^{k,-} \sim P(\boldsymbol{X}_0)$ and $w_0^{k,-} = 1$.

Prediction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_{i-1})$ by

$$\left\{(\boldsymbol{s}_i^{k,-}, w_i^{k,-})\right\}$$

where

$$\begin{aligned} \boldsymbol{s}_i^{k,-} &= f(\boldsymbol{s}_{i-1}^{k,+}) + \xi_i^k \\ w_i^{k,-} &= w_{i-1}^{k,+} \end{aligned}$$

and

$$\xi_i^k \sim N(0, \Sigma_{d_i}$$

Correction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_i)$ by

$$\left\{(\boldsymbol{s}_i^{k,+}, w_i^{k,+})\right\}$$

where

$$\begin{aligned} \boldsymbol{s}_i^{k,+} &= \boldsymbol{s}_i^{k,-} \\ w_i^{k,+} &= P(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-})w_i^{k,-} \end{aligned}$$

Resampling: Normalise the weights so that $\sum_i w_i^{k,+} = 1$ and compute the variance of the normalised weights. If this variance exceeds some threshold, then construct a new set of samples by drawing, with replacement, N samples from the old set, using the weights as the probability that a sample will be drawn. The weight of each sample is now $1/N$.

Algorithm 18.8: *A practical particle filter resamples the posterior.*

Initialization: Represent $P(\boldsymbol{X}_0)$ by a set of N samples

$$\left\{(\boldsymbol{s}_0^{k,-}, w_0^{k,-})\right\}$$

where

$$\boldsymbol{s}_0^{k,-} \sim P_s(\boldsymbol{S})$$

and

$$w_0^{k,-} = P(\boldsymbol{s}_0^{k,-})/P_s(\boldsymbol{S} = \boldsymbol{s}_0^{k,-})$$

Ideally, $P(\boldsymbol{X}_0)$ has a simple form and $\boldsymbol{s}_0^{k,-} \sim P(\boldsymbol{X}_0)$ and $w_0^{k,-} = 1$.

Prediction: Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_{i-1})$ by

$$\left\{(\boldsymbol{s}_i^{k,-}, w_i^{k,-})\right\}$$

where

$$\begin{aligned} \boldsymbol{s}_i^{k,l,-} &= f(\boldsymbol{s}_{i-1}^{k,+}) + \xi_i^l \\ w_i^{k,l,-} &= w_{i-1}^{k,+} \end{aligned}$$

and

$$\xi_i^l \sim N(0, \Sigma_{d_i}$$

and the free index l indicates that each $\boldsymbol{s}_{i-1}^{k,+}$ generates M different values of $\boldsymbol{s}_i^{k,l,-}$. This means that there are now MN particles.

Correction: We reindex the set of MN samples by k. Represent $P(\boldsymbol{X}_i|\boldsymbol{y}_0, \boldsymbol{y}_i)$ by

$$\left\{(\boldsymbol{s}_i^{k,+}, w_i^{k,+})\right\}$$

where

$$\begin{aligned} \boldsymbol{s}_i^{k,+} &= \boldsymbol{s}_i^{k,-} \\ w_i^{k,+} &= P(\boldsymbol{Y}_i = \boldsymbol{y}_i|\boldsymbol{X}_i = \boldsymbol{s}_i^{k,-})w_i^{k,-} \end{aligned}$$

Resampling: As in algorithm 8.

Algorithm 18.9: *An alternative practical particle filter.*

a sampled representation is the true value of the integral, it may require a very large number of particles before the variance of the estimator is low enough to be acceptable. It is difficult to say how many particles will be required to produce

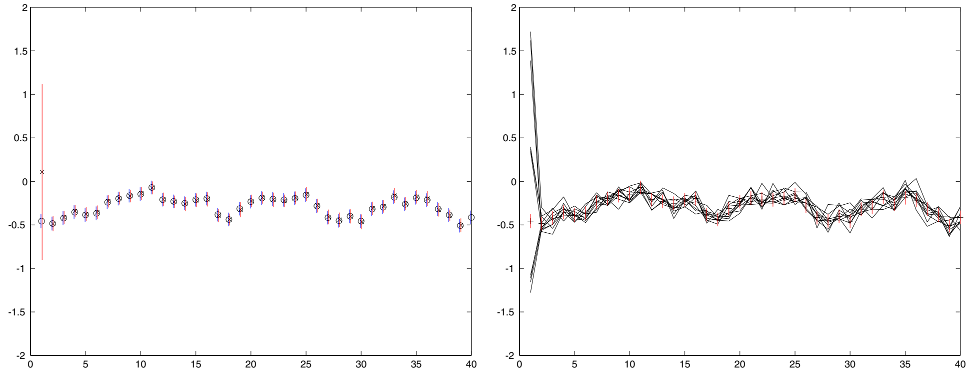


Figure 18.10. Resampling hugely improves the behaviour of a particle filter. We now show a resampled particle filter tracking a point drifting on the line (i.e. $x_i \sim N(x_{i-1}, \sigma^2)$). The measurements are corrupted by additive Gaussian noise, and are the same as for figure 18.9. In the figure on the **left**, we compare an exact representation obtained using a Kalman filter with one computed from simple particle filtering. We show the mean of the posterior as a point with a one standard deviation bar. The mean obtained using a Kalman filter is given as an 'x'; the mean obtained using a particle filter is given as an 'o'; we have offset the standard deviation bars from one another so as to make the phenomenon clear. Notice that estimates of both mean and standard deviation obtained from the particle filter compare well with the exact values obtained from the Kalman filter. The figure on the **right** indicates where this improvement came from; we plot the tracks of ten particles, randomly selected from the 100 used. Because we are now resampling the particles according to their weights, particles that tend to reflect the state rather well usually reappear in the resampled set. This means that many particles lie within one standard deviation of the mean of the posterior, and so the weights on the particles tend to have much smaller variance, meaning the representation is more efficient.

usable estimates. In practice, this problem is usually solved by experiment.

Unfortunately, these experiments may be misleading. You can (and should!) think about a particle filter as a form of search — we have a series of estimates of state, which we update using the dynamic model, and then compare to the data; estimates which look as though they could have yielded the data are kept, and the others are discarded. The difficulty is that we may miss good hypotheses. This could occur if, for example, the likelihood function had many narrow peaks. We may end up with updated estimates of state that lie in some, but not all of these peaks; this would result in good state hypotheses being missed. While this problem can (just!) be caused to occur in one dimension, it is particularly serious in high dimensions. This is because real likelihood functions can have many peaks, and these peaks are easy to miss in high dimensional spaces. It is extremely difficult to get good results from particle filters in spaces of dimension much greater than about 10.

The problem can be significant in low dimensions, too — its significance depends, essentially, on how good a prediction of the likelihood we can make. This problem manifests itself in the best-known fashion when one uses a particle filter to track people. Because there tend to be many image regions that are long, roughly straight, and coherent, it is relatively easy to obtain many narrow peaks in the likelihood function — these correspond, essentially, to cases where the configuration for which the likelihood is being evaluated has a segment lying over one of these long, straight coherent image regions. While there are several tricks for addressing this problem — all involve refining some form of search over the likelihood — there is no standard solution yet.

18.5 Data Association

Not every aspect of every measurement conveys information about the state of the object being tracked. In fact, we have been somewhat disingenuous up to this point, and not really talked about what is in $\boldsymbol{y}_i$ at all. Usually, there are measurements that are informative and measurements that are not informative (usually referred to as **clutter**).

Determining which measurements are informative is usually referred to as **data association**. Typically, one wishes to map a series of measurements to a series of tracks. The main work in this problem relates to tracking moving objects (aeroplanes, missiles, etc., all conveniently belonging to the bad guys) with radar returns. Typically, there may be many radar returns at any given timestep — we should like to update our representations of the motion of the objects being tracked without necessarily knowing which returns come from which object. As we have seen, tracking algorithms are complicated, but not particularly difficult. Data association is probably the biggest source of difficulties in vision applications. We will confine our discussion to the case where there is a single moving object. The problem here is that some pixels in the image are very informative about that object, and some are not — which should we use to guide our tracking process?

18.5.1 Choosing the Nearest — Global Nearest Neighbours

In the easiest case, we need to track a single object moving in clutter. For example, we might be tracking a ball moving on a fixed or very slowly varying background. We segment the image into regions, with the reasonable expectation that the ball tends to produce one region, and that the segmentation of the background might change with time. Intuitively, it would be very difficult to confuse the ball with a background region, because we have a strong model of how the ball is moving. This means we would have to be unlucky if there was a new background region that (a) was easily confused with the ball region and (b) confused the dynamic model. This suggests one fairly popular strategy for data association: the r'th region offers a measurement $\boldsymbol{y}_i^r$, and we choose the region with the best value of

$$P(\boldsymbol{Y}_i = \boldsymbol{y}_i^r | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) = \int P(\boldsymbol{Y}_i = \boldsymbol{y}_i^r | \boldsymbol{X}_i, \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) d\boldsymbol{X}_i$$

$$= \int P(\boldsymbol{Y}_i = \boldsymbol{y}_i^r | \boldsymbol{X}_i) P(\boldsymbol{X}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) d\boldsymbol{x}_i$$

Determining $P(\boldsymbol{Y}_i = \boldsymbol{y}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ is a particularly easy calculation with the Kalman filter. We know how $\boldsymbol{Y}_i$ is obtained from $\boldsymbol{X}_i$ — we take a normal random variable with mean $\overline{\boldsymbol{X}}_i^-$, and covariance Σ_i^-, apply the linear operator $\mathcal{D}_i$ to it, and add some other random variable. The output of the linear operator must have mean $\mathcal{D}_i \overline{\boldsymbol{X}}_i^-$ and covariance $\mathcal{D}_i \Sigma_i^- \mathcal{D}^T$. To this we are going to add a random variable with zero mean and covariance Σ_{m_i}; the result must have mean

$$\mathcal{D}_i \overline{\boldsymbol{X}}_i^-$$

and covariance

$$\mathcal{D}_i \Sigma_i^- \mathcal{D}^T + \Sigma_{m_i}$$

In figure **??**, we have plotted bounds on the position of an expected measurement for a Kalman filter following various dynamic models.

Notice that this strategy can be relatively robust, depending on the accuracy of the dynamic model. If we have a good dynamic model, anything that is easily confused with the object being tracked must be more similar to the predicted measurement than the real object does. This means that an occasional misidentification may not create major problems, because one is unlikely to find a region that is both similar to the predicted measurement and able to throw off the dynamic model badly. In figure 18.12, we show a Kalman filter tracking the state of a point by choosing the best measurement at each step; it does not always correctly identify the point, but its estimate of state is always good.

Notice that what we are doing here is using only measurements that are consistent with our predictions. This may or may not be dangerous: it can be very easy to track non-existent objects this way — or to claim to be tracking an object without ever obtaining a measurement from it. If the dynamic model itself can give only weak predictions — i.e. the object doesn't really behave like that, or Σ_{d_i} is consistently large — we may have serious problems, because we will need to rely on the measurements. These problems occur because the error can accumulate — it is now relatively easy to continue tracking the wrong point for a long time, and the longer we do this the less chance there is of recovering the right point. Figure 18.13 shows a Kalman filter becoming hopelessly confused in this manner.

18.5.2 Gating and Probabilistic Data Association

Again, we assume that we are tracking a single object in clutter, and use the example of tracking a ball moving on a fixed or very slowly varying background. Instead of choosing the region most like the predicted measurement, we could exclude all regions that are too different, and then use all others, weighting them according to their similarity to the prediction.

The first step is called **gating**. We exclude all measurements that are too different from the predicted measurement. What "too different" means rather depends

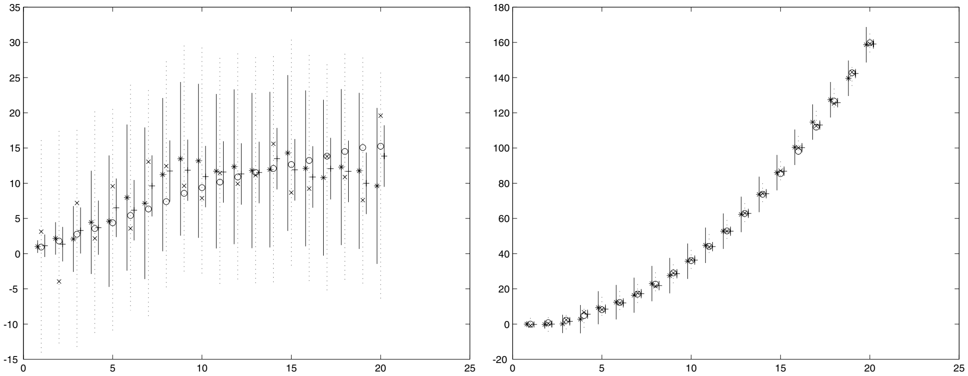


Figure 18.11. Data association for a Kalman filter for a point moving on the line under our model of constant velocity on the **left** and constant acceleration on the **right**. Compare with figure 18.1 for the constant velocity model and with figure 18.2 for the constant acceleration model. We have used the conventions of figure **??**. We have now overlaid 3 standard deviation bars for the measurement (the dashed bars passing through the state). These are obtained using the estimate of state before a measurement, and our knowledge of the variance of the measurement process. Notice that the measurements lie within these windows.

on the application: if we are too aggressive in excluding measurements, we may find nothing left. It is usual to exclude measurements which lie more than some number — commonly, three — of standard deviations from the predicted mean. A more sophisticated strategy is required if the object being tracked has more than one dynamic behaviour; for example, military aircraft often engage in high-speed maneuvers. In cases like this, it is common to have several gates, and to take all measurements that lie in the tightest gate that contains any measurements.

The next step is called **probabilistic data association**, usually abbreviated PDA. Assume that, in the gate, we have a set of N regions, each producing a vector of measurements $\boldsymbol{y}_i^k$, where the superscript indicates the region. We have a set of possible hypotheses to deal with: either no region comes from the object, which we shall call h_0, or region k comes from the object, which we shall call h_k. The measurement we report is

$$\mathrm{E}_h\left[\boldsymbol{y}_i\right] = \sum_j P(h_j|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})\boldsymbol{y}_i^j$$

where the expectation is taken over the space of hypotheses (which is why we have given it the subscript h). Now the probability that none of the measurements comes from the object depends on the details of our detection process. For some detection processes, this parameter can be calculated; for example, in chapter 4 of [], there is a worked example for a radar system. In other cases we will need to search for a value of the parameter that results in good behaviour on a set of training

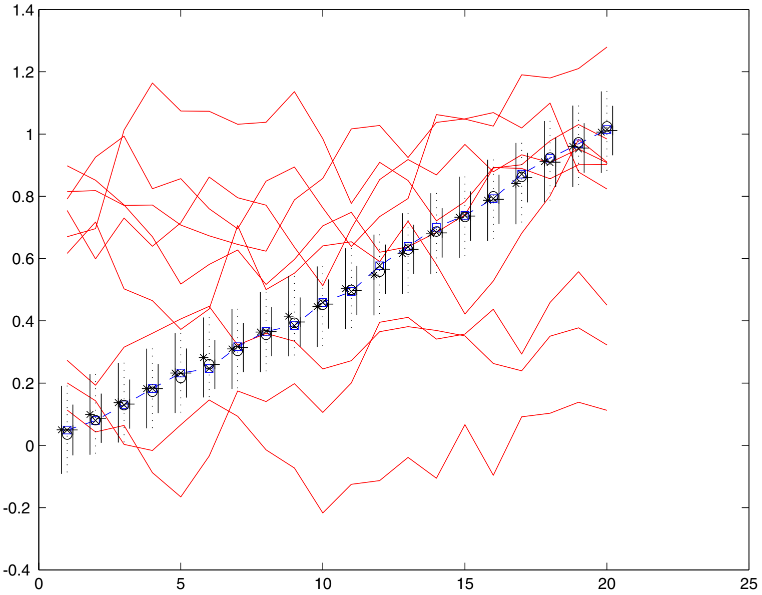


Figure 18.12. Predictions of the point position can identify "good" measurements for a Kalman filter. We are using a Kalman filter to identify a point, moving with constant velocity on a line, and with a small Σ_{d_i} at every stage. There are also 10 drifting points. This plot shows position plotted against time for the drifting points — which are shown with a solid line — and for the point that is being tracked. The trajectory of the point that should be tracked is shown in a dashed line, and each measurement on this trajectory is shown with a square. We have used the conventions of figure **??** (i.e. the state is plotted with open circles, as a function of the step i; the '*'-s give $\overline{\boldsymbol{x}}_i^-$, which is plotted slightly to the left of the state to indicate that the estimate is made before the measurement; the 'x'-s give the measurements, and the '+'-s give $\overline{\boldsymbol{x}}_i^+$, which is plotted slightly to the right of the state; the vertical bars around the '*'-s and the '+'-s are 3 standard deviation bars, using the estimate of variance obtained before and after the measurement, respectively; we have overlaid one standard deviation bars in each case). This filter chooses the measurement at each step by choosing the measurement that maximizes $P(\boldsymbol{y}_i^r|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$; notice that it doesn't choose the right measurement at every step (i.e. the x is not always in the square), but it maintains a very good estimate of the state (i.e. the +'s are close to the circles).

examples. Assume that we have calculated or learned this parameter, which we can write as β. We must also assume that either the object is not detected, or only one measurement comes from the object. Now

$$\begin{aligned} P(h_j|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) &= \int P(h_j|\boldsymbol{X}_i) P(\boldsymbol{X}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) d\boldsymbol{X}_i \\ &= P(\boldsymbol{Y}_i = \boldsymbol{y}_i^j|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) P(\text{object detected}) \end{aligned}$$

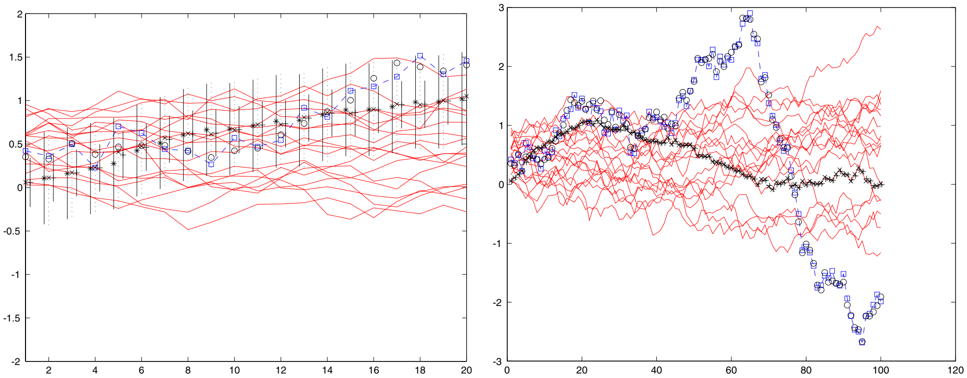


Figure 18.13. If the dynamic model is not sufficiently constrained, then choosing the measurement that gives the best $P(\boldsymbol{y}_i^r|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ can lead to disaster. On the **left**, 20 steps of a Kalman filter following a point moving periodically on the line with 20 drifting points in the background. We are using the conventions of figure **??** again. Now Σ_{d_i} is relatively large for each step, and so it is easy to follow the wrong measurement for some way. It looks as though the filter is tracking the state well, but in fact as the figure on the **right** — which gives 100 steps— shows, it quickly becomes hopelessly lost.

$$= P(\boldsymbol{Y}_i = \boldsymbol{y}_i^j|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})(1-\beta)$$

(in what follows, we write $P(h_j|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ as p_j). In practice, this method is usually used with a Kalman filter. To do so, we report the measurement

$$\boldsymbol{y}_i' = \sum_j p_j \boldsymbol{y}_i^j$$

to the Kalman update equations. Note that the term for not having a measurement appears here as the factor $(1-\beta)$ in the expressions for the p_j; but our uncertainty about which measurement should contribute to the update should also appear in the covariance update. We modify the covariance update equations to take the form

$$\Sigma_i^+ = (1-\beta)\left[Id - \mathcal{K}_i\mathcal{M}_i\right]\Sigma_i^- + \beta\Sigma_i^- + \mathcal{K}_i\left[\sum_j p_j(\mathcal{H}_i\overline{\boldsymbol{x}}_i^- - \boldsymbol{y}_i^j)(\mathcal{H}_i\overline{\boldsymbol{x}}_i^- - \boldsymbol{y}_i^j)^T - \boldsymbol{y}_i'(\boldsymbol{y}_i')^T\right]$$

Here the first term is the update for the standard Kalman filter weighted by the probability that one observation is good, the second term deals with the prospect that all observations are bad, and the third term contributes uncertainty due to the correspondence uncertainty.

18.6 Applications and Examples

Tracking is a technology with a number of possible applications. There are three dominant topics.

- **Vehicle tracking** systems could report traffic congestion, accidents and dangerous or illegal behaviour by road users. Traffic congestion reports are useful for potential road users — who might change their travel plans — and to authorities — who might arrange to remove immobilised vehicles blocking lanes, etc. Accident reports can be used to alert emergency services; if the tracking system can read vehicle numberplates, it might use reports of dangerous or illegal behaviour to send a summons to the vehicle owner.

- **Surveillance** systems report what people are doing, usually with the aim of catching people who are doing things they shouldn't. The police might wish to know which member of a sports audience threw a bottle onto the field, for example; or if the same person visited several different banks just before they were robbed. Customs might wish to know exactly who is loading and unloading aircraft flying to foreign ports.

- **Human-Computer interaction** systems use people's actions to drive various devices. For example, the living room might decide for itself, by watching what people are doing, when low lights and soft music are appropriate. The television set might change channels when you wave at it. Your computer might watch what you write on your whiteboard, and make a record of the contents when you tell it to.

Currently, the most convincing applications are in vehicle tracking. These systems work reliably under a large range of circumstances. We shall survey vehicle tracking systems briefly, and then sketch some work in surveillance and hci. These systems tend to be rather less robust — most systems come with a rich collection of constraints that must be true before they can be used. This is because people have a large number of degrees of freedom: bits of the body move around, we can change clothing, etc., which means it is quite difficult to predict appearance.

18.6.1 Vehicle Tracking

Systems that can track cars using video from fixed cameras can be used to predict traffic volume and flow; the ideal is to report on, and act to prevent, traffic problems as quickly as possible. A number of systems can track vehicles successfully. The crucial issue is initiating a track automatically. In the two systems we describe here, the problem is attacked quite differently. Sullivan *et al.* construct a set of regions of interest (ROI's) in each frame. Because the camera is fixed, these regions of interest can be chosen to span each lane (figure 18.14); this means that almost all vehicles must pass directly through a region of interest in a known direction (there are mild issues if a vehicle chooses to change lanes while in the ROI, but these can be ignored). Their system then watches for characteristic edge signatures in the ROI that indicate the presence of a vehicle (figure 18.14). These signatures can alias slightly — typically, a track is initiated when the front of the vehicle enters the ROI, another is initiated when the vehicle lies in the ROI, and a third

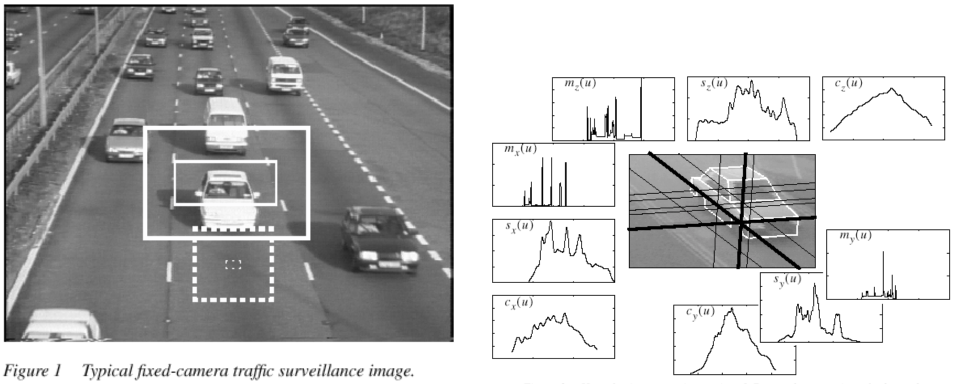


Figure 1 *Typical fixed-camera traffic surveillance image. The position of two "traps" are shown:*
(i) Hypothesis generation (solid boxes)
(ii) Hypothesis verification (dashed boxes)

Figure 2 *Hypothesis generation, using 1-D templates projected along the three vehicle-centred axes. The triples of distributions show: model projections* (m)*, data profiles* (s) *and their correlations* (c)
Black lines show the main peaks of c *back-projected along the axes.*

Figure 18.14. Sullivan *et al.* track vehicles in views of the road from a stationary camera. Their tracker uses a series of regions of interest registered to the road, which are shown on the **left**. They initiate tracks by looking for characteristic edge signatures in a particular ROI; these signatures are projected onto three distinct coordinate axes — if the edges projected on these axes have a high enough correlation with the expected form, then a track is initiated (**right**). *figure from Model-based vehicle detection and classification using orthographic approximations, G D Sullivan, K D Baker, A D Worrall, C I Attwood and P R Remagnino p.2, 4, in the fervent hope of receiving permission*

is initiated close to the vehicle's leaving — because some of the vehicle's edges are easily mistaken for others.

Each initiated track is tracked for a sequence of frames, during which time it accumulates a quality score — essentially, an estimate of the extent to which predictions of future position were accurate. If this quality score is sufficiently high, the track is accepted as an hypothesis. An exclusion region in space and time is constructed around each hypothesis, such that there can be only one track in this region, and if the regions overlap, the track with the highest quality is chosen. The requirement that the exclusion regions do not overlap derives from the fact that two cars can't occupy the same region of space at the same time. Once a track has passed these tests, the position in which and the time at which it will pass through another ROI can be predicted. The track is finally confirmed or rejected by comparing this ROI at the appropriate time with a template that predicts the car's appearance. Typically, relatively few tracks that are initiated reach this stage (figure 18.15).

An alternative method for initiating car tracks is to track individual features, and then group those tracks into possible cars. Beymer *et al.* use this strategy rather successfully. Because the road is plane and the camera is fixed, the homography connecting the road plane and the camera can be determined. This homography can be used to determine the distance between points; and points can lie together

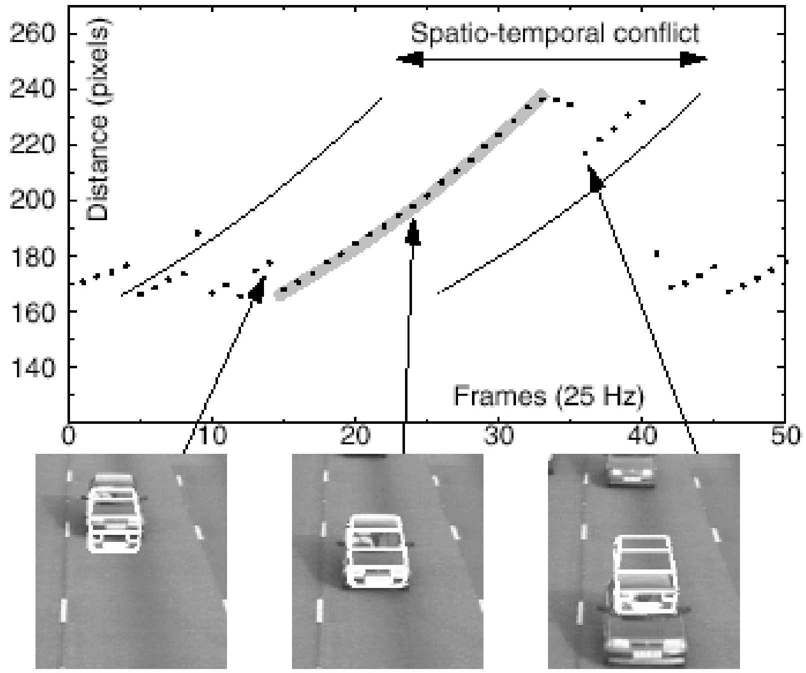


Figure 3 Strongest hypotheses (for one model), shown as position in the image in the direction of travel (ordinate), against frame number (abscissa). Note how "aliases" have short tracks, and become superseded by the "true" hypotheses (grey bar) - see text.

Figure 18.15. In the system of Sullivan *et al.*, tracks are continued if they are of sufficient quality, measured by comparing the prediction of the track with measurements. Tracks exclude other tracks: by the time a car reaches the bottom of the view, the system must decide which track to accept. It does so by comparing the track prediction with another ROI. This figure plots a series of tracks (position on the vertical axis and time on the horizontal axis). Notice that the typical alias tracks (that arise because the front of a car and the back of a car both tend to look rather like a registered car to the track initiation process) tend to die out quite quickly; the real track, and its exclusion regions, is indicated. If two tracks attempt to exclude one another, the winner is the track of the highest quality. *figure from Model-based vehicle detection and classification using orthographic approximations, G D Sullivan, K D Baker, A D Worrall, C I Attwood and P R Remagnino p.5, in the fervent hope of receiving permission*

on a car only if this distance doesn't change with time. Their system tracks corner points, identified using a second moment matrix (section **??**), using a Kalman filter. Points are grouped using a simple algorithm using a graph abstraction: each feature track is a vertex, and edges represent a grouping relationship between the tracks.

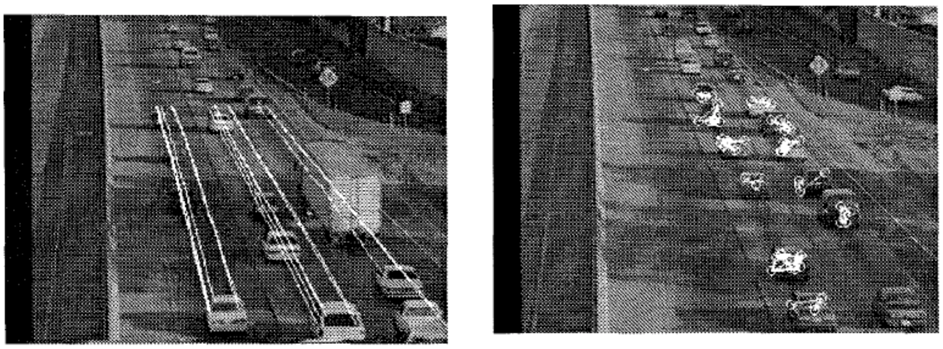


Figure 4: Example tracks of corner features.

Figure 5: Example groups of corner features.

Figure 18.16. The figure on the **left** shows individual tracks for the system of Beymer et al. These tracks are obtained by tracking corner points with a Kalman filter. Because the camera position with respect to the road plane is known, the camera transformation can be inverted for points lying on a plane parallel to the road plane. This means that we can determine pairs of points that remain at a constant distance from one another. The figure on the **right** shows groups of such points. These groups are assumed to represent vehicles. *figure from A Real-Time Computer Vision System for Measuring Traffic Parameters, Beymer, McClachlan, Coifman and Malik et al. p.498, in the fervent hope of receiving permission*

When a new feature comes into view — and a track is thereby initiated — it is given an edge joining it to every feature track that appears nearby in that frame. If, at some future time, the distance between points in a track changes by too much, the edge is discarded. An exit region is defined near where vehicles will leave the frame. When tracks reach this exit region, connected components are defined to be vehicles. This grouper is successful, both in example images (figure 18.16) and in estimating traffic parameters over long sequences (figure 18.17).

The ground plane to camera transformation can provide a great deal of information; once an object has been tracked, we can use this transformation to reason about spatial layout and occlusion. Remagnino *et al.* track vehicles and pedestrians — pedestrians in coarse scale images are represented with a closed B-spline curve, whose control points are tracked with a Kalman filter; the B-spline tracks edge data, using a fairly narrow gate around a set of discrete points along the spline —- and then reconstruct spatial relations using this homography. The advantage of this approach is that one can engage in explicit occlusion reasoning, so that even pedestrians partially occluded by a car can be tracked (figure 18.18).

Another use of the homography makes it possible to track cars from moving vehicles. In this case, there are two issues to manage: firstly, the motion of the camera platform (so-called **ego-motion**); and secondly, the motion of other vehi-

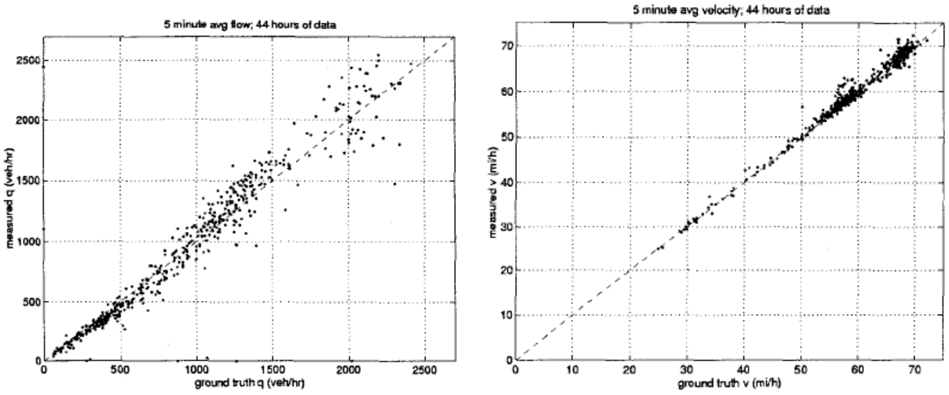


Figure 18.17. The system of Beymer et al. can produce rather accurate estimates of traffic flow and traffic velocity. On the **left** a scatter plot of estimates of flow vs. ground truth, and on the **right** a scatter plot of estimates of velocity vs. ground truth. *figure from A Real-Time Computer Vision System for Measuring Traffic Parameters, Beymer, McClachlan, Coifman and Malik et al. p.500, in the fervent hope of receiving permission*

cles. Maybank *et al.* estimate the ego-motion by matching views of the road to one another from frame to frame (figure 18.19). With an estimate of the homography and of the ego-motion, we can now refer tracks of other moving vehicles into the road coordinate system to come up with reconstructions of all vehicles visible on the road from a moving vehicle (figure 18.19).

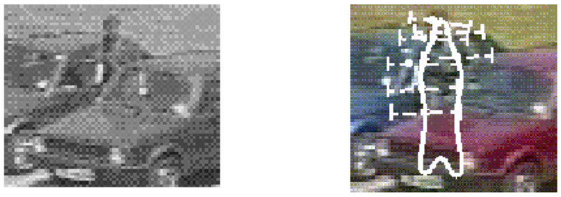
18.6.2 Finding and Tracking People

Tracking people is difficult. The first difficulty is that there is a great deal of state to a human — there are many joint angles, etc. that may need to be represented. The second difficulty is that it is currently very hard to find people in an image — this means that it can be hard to initiate tracks.

Tracking People

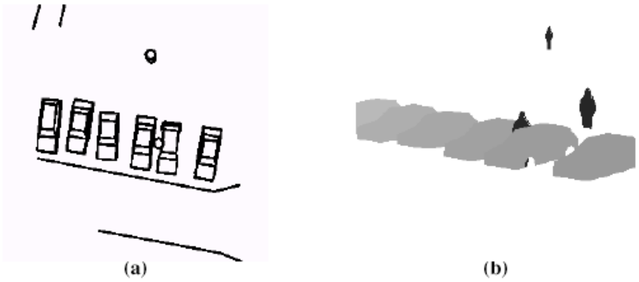
People are typically modelled as a collection of body segments, connected with rigid transformations. These segments can be modelled as cylinders — in which case, we can ignore the top and bottom of the cylinder and any variations in view, and represent the cylinder as an image rectangle of fixed size — or as ellipsoids. The state of the tracker is then given by the rigid body transformations connecting these body segments (and perhaps, various velocities and accelerations associated with them).

Both particle filters and (variants of) Kalman filters have been used to track people. Each approach can be made to succeed, but neither is particularly robust. There are two components to building a particle filter tracker: firstly, we need a



(a) (b)

Figure 4: (a) Driver gets out of car, (b) Shape mode overlaid with unoccluded search lines.



(a) **(b)**

Figure 2: (a) Ground plane map used to solve occlusions, (b) Example of silhouette image.

Figure 18.18. The figure on the **top** illustrates Remagnino *et al.*'s representation of pedestrians in their tracker; the pedestrians are represented as a simple closed B-spline curve whose control points are tracked. Each curve carries a set of gates and search directions, which are used for data association — the nearest edge point in the search direction is reported as the new data point. The control points of the curves are tracked with a Kalman filter. On the **bottom**, 3D representations obtained using their tracker; firstly, one can reconstruct an overhead view of cars and pedestrians (**left**) and secondly, one can construct a depth representation that makes it possible to evaluate occlusion relationships sensibly (**right**). *figure from An Integrated Traffic and Pedestrian Model-Based Vision System P.Remagnino 1 , A.Baumberg 2 , T.Grove 1 , D.Hogg 2 , T.Tan 1 , A.Worrall 1 , K.Baker 1 p.8,9, in the fervent hope of receiving permission*

motion model and secondly, we need a likelihood model.

We can use either a strong motion model — which can be obtained by attaching markers to a model and using them to measure the way the model's joint angles change as a function of time — or a weak motion model — perhaps a drift model. Strong motion models have some disadvantages: perhaps the individual we are tracking moves in a funny way; and we will need different models for walking, walking carrying a weight, jogging and running (say). The difficulty with a weak motion model is that we are pretty much explicitly acknowledging that each frame is a poor guide to the next.

Likelihood models are another source of difficulties, because of the complexity of the relationship between the tracker's state and the image. The likelihood func-

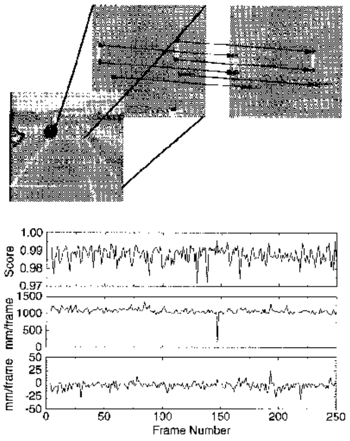


Figure 2. Egomotion estimation: (a) projection of image region onto ground-plane with detected features and corresponding located features in the warped image of the next frame; and (b) recovered motion parameters.

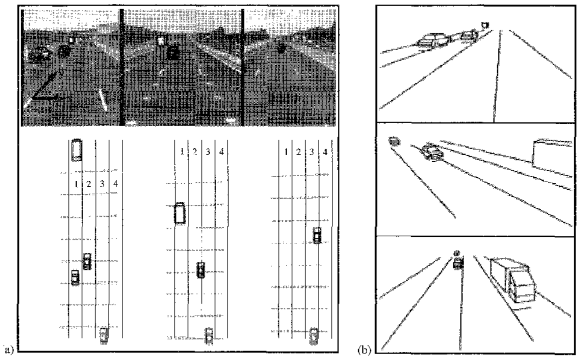


Figure 18.19. On the **left**, the ego-motion estimation of Maybank *et al.*, which compares distinct views of the ground-plane to obtain an estimate of motion. Once the motion — and hence position, allowing for an arbitrary choice of origin — has been estimated, tracks of other vehicles obtained using the moving camera platform can be referred to the coordinate system on this ground plane. This allows detailed reconstructions of traffic geometry, illustrated on the **right**; such reconstructions can be used, for example, to predict impending collision between the camera platform and some other vehicle. *figure from Visual Surveillance for moving vehicles Ferryman, Maybank and Worrall p.75,76, in the fervent hope of receiving permission*

tion ($P(\text{image features}|\text{person present at given configuration})$) tends to have many local extrema. This is because the likelihood function is evaluated by, in essence, rendering a person using the state of the tracker and then comparing this rendering to the image. Assume that we know the configuration of the person in the previous image; to assess the likelihood of a particular configuration in the current image, we use the configuration to compute a correspondence between pixels in the current image and in the previous image. The simplest likelihood function can be obtained using the sum of squared differences between corresponding pixel values — this assumes that clothing is rigid with respect to the human body, that pixel values are independent given the configuration, and that there are no shading variations. These are all extremely dubious assumptions.

Of course, we choose which aspects of an image to render and to compare; we might use edge points instead of pixel values, to avoid problems with illumination. Multiple extrema in the likelihood can be caused by: the presence of many extended

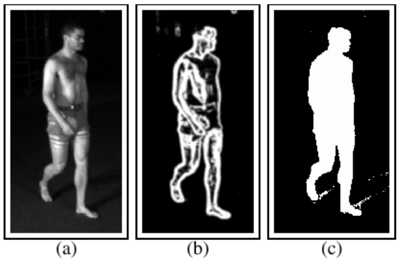


Figure 5: **Feature extraction.** A gradient based edge detection mask is used to find edges. The result is thresholded to eliminate spurious edges and smoothed using a Gaussian mask to produce a pixel map (b) in which the value of each pixel is related to it proximity to an edge. The foreground is segmented using thresholded background subtraction to produce the pixel map (c) used in the weighting function.

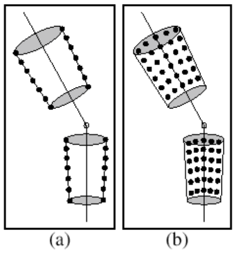


Figure 6: **Configurations of the pixel map sampling points** $p_i(\mathbf{X},\mathbf{Z})$ for the edge based measurements (a) and the foreground segmentation measurements (b). The sampling points for the edge measurements are located along the occluding contours of the model's conical sections that have been projected into the image. The sampling points for the foreground segmentation measurements are taken from a grid within these occluding contours.

Figure 18.20. A typical likelihood computation identifies points in the image that provide evidence that a person is present. In one use of a particle filter for tracking people, Deutscher, Blake and Reid look for two types of evidence: the first is boundary information, and the second is "non-background" information. Boundary points are estimated using an edge detector, and "non-background" points are obtained using background subtraction; the figure on the **left** illustrates these types of point. Now each particle gives the state of the person, and so can be used to determine where each body segment lies in an image; this means we can predict the boundaries of the segments and their interiors, and compute a score based on the number of edge points near segment boundaries and the number of "non-background" points inside projected segments (**right**). *figure from Articulated Body Motion Capture by Annealed Particle Filtering, Jonathan Deutscher and Andrew Blake and Ian Reid p.6, 6, in the fervent hope of receiving permission*

coherent regions, which look like body segments, in images; the presence of many edge points unrelated to the person being tracked (this is a problem if we use edge points in the comparison); changes in illumination; and changes in the appearance of body segments caused by clothing swinging on the body. The result is a tendency for trackers to drift (see, for example, the conclusions in [**?**]; the comments in [**?**]).

Figure 18.21 illustrates a human tracker that uses a strong motion model and particle filtering to track a person. In all of the examples we show, the tracker must be started by hand. An alternative to using a strong motion model is to use a weak motion model and rely on the search component of particle filtering. The best example of this approach is a device known as an annealed particle filter (figure **??**), which essentially searches for the global extremum through a sequence of smoothed versions of the likelihood [**?**]. However, clutter creates peaks that can require intractable numbers of particles. Furthermore, this strategy requires a detailed search of a high dimensional domain (the number of people being tracked times the number of parameters in the person model plus some camera parameters).

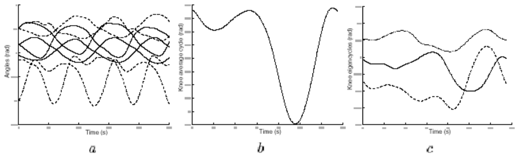


Fig. 3. Learning a walking model. *(a)* Joint angles of different people walking were acquired with a motion capture system. Curves are segmented into walking cycles manually and an eigenmodel of the cycle is constructed. *(b)* Mean angle of left knee as a function of time. *(c)* First three eigenmodes of the left knee $\mathbf{B}_j$, $j \in [1, 3]$, scaled by their respective variance λ_j. ($1 = solid, 2 = --, 3 = \cdots$)

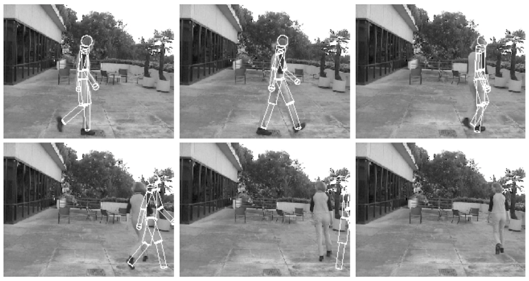


Fig. 7. How strong is the walking prior? Tracking results for frames 0, 10, 20, 30, 40 and 50, when no image information is taken into account.

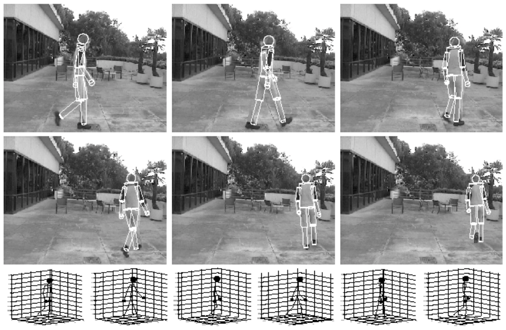


Fig. 6. Person walking in a circle (15000 samples). Upper rows: frames 0, 10, 20, 30, 40, 50 with the projection of the expected model configuration overlaid. Lower row: expected 3D configuration in the same frames.

Figure 18.21. A particle filter can be used to track a person, represented as a set of plane rectangles, each standing for a body segment. The top figure shows a representation of the joint angles of a human walker plotted as a function of time. These are used to give $P(\boldsymbol{X}_i|\boldsymbol{X}_{i-1})$. The center figure shows a human walker being tracked. The dynamic model is extremely strong; the lower figure shows a human walker being tracked *in the absence of image evidence.* Notice that the state of the tracker eventually drifts from the correct state — it can't tell when the person has turned — but that this takes time, because the dynamic model is so detailed. *figure from Stochastic Tracking of 3D Human Figures Using 2D Image Motion, Hedvig Sidenbladh and Michael J. Black and David J. Fleet, p.11,16,17, in the fervent hope of receiving permission*

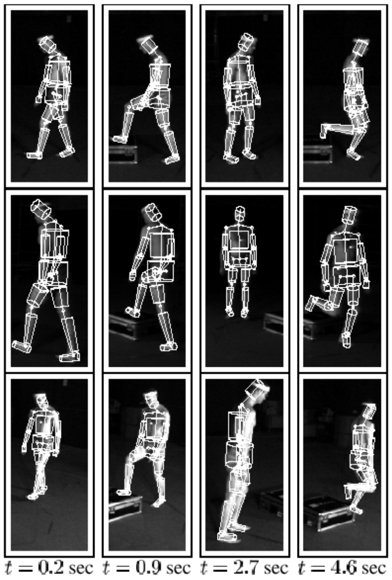


Figure 10: **Stepping over a box.** Using three cameras (arrayed here from top to bottom) a person is tracked over 5 seconds while stepping over a box, turning around and stepping over the box again. The tracker maintains an accurate lock throughout. 10 annealing layers were used with 200 particles for this sequence. Download the movie from *www.robots.ox.ac.uk/~ jdeutsch/HMC/.*

Figure 18.22. If a weak motion model is used to track a person with a particle filter, the likelihood function can create serious problems. This is because the state is high-dimensional, and there are many local peaks in the likelihood — even for a person on a black background, as in these images. It is quite possible that none of our particles are near the peaks, meaning that the filter's representation becomes unreliable. One way to deal with this is to search the likelihood by *annealing* it. This process creates a series of increasingly smooth approximations to the likelihood, whose peaks lie close to or on the peaks of the likelihood. We weight particles with a smoothed approximation, then resample the particles according to their weights, allow them to drift, then weight them with a less smooth approximation, etc. The result is a random search through the likelihood that should turn up the main local minima. This yields a tracker that can track people on simple backgrounds, but requires only very general motion models — the tracker illustrated above models human motion as drift. *figure from Articulated Body Motion Capture by Annealed Particle Filtering, Jonathan Deutscher and Andrew Blake and Ian Reid p.8, in the fervent hope of receiving permission*

Finding People

There is currently (Oct, 2000) no person tracker that represents the configuration of the body and can start automatically; all such trackers use manual starting methods. One way to start such a tracker would be to find all possible people, and then track them. But finding people is difficult, too. No published method can find clothed people in arbitrary configurations in complex images. There are three standard approaches to finding people described in the literature. Firstly, the problem can be attacked by template matching (section **??**; examples include [**?**], where upright pedestrians with arms hanging at their side are detected by a template matcher; [**?**; **?**; **?**], where walking is detected by the simple periodic structure that it generates in a motion sequence; [**?**; **?**], which rely on background subtraction — that is, a template that describes "non-people"). Matching templates to people (rather than to the background) is inappropriate if people are going to appear in multiple configurations, because the number of templates required is too high. This motivates the second approach, which is to find people by finding faces (section **??**, and [**?**; **?**; **?**; **?**; **?**; **?**]). The approach is most successful when frontal faces are visible.

The third approach is to use the classical technique of search over correspondence (search over correspondences between point features is an important early formulation of object recognition; the techniques we describe have roots in [**?**; **?**; **?**; **?**]). In this approach, we search over correspondence between image configurations and object features. There are a variety of examples in the literature (for a variety of types of object; see, for example, [**?**; **?**]). Perona and collaborators find faces by searching for correspondences between eyes, nose and mouth and image data, using a search controlled by probabilistic considerations [**?**; **?**]. Unclad people are found by [**?**; **?**], using a correspondence search between image segments and body segments, tested against human kinematic constraints. A much improved version of this technique, which learns the model from data, appears in [**?**]. Song *et al.* formulate finding people in moving light displays as a correspondence search through a probabilistic model and use dynamic programming to solve the search.

18.7 Discussion

We have been able to provide only a brief overview of a subject that is currently extremely active. We have deliberately phrased our discussion rather abstractly, so as to bring out the issues that are most problematic, and to motivate a view of extended Kalman filters and particle filters as convenient approximations.

Particle filters have been the subject of a great deal of work in vision. Much of the work attempts to sidestep the difficulties with likelihood functions that we sketched in the particle filtering section. Unfortunately, all uses of the particle filter have been relentlessly top-down — in the sense that one updates an estimate of state and then computes some form of difference between an image and a rendering, which is asserted to be a likelihood. This is certainly not an intrinsic property of the technique — which is just an algorithm — and is almost certainly a major strategic error. The consequence of this error is that one can track almost anything with a particle filter (as long as the dimension of the state space is small enough) but it has to be initialized by hand. This particular ghost needs to be exorcised from the party as soon as possible.

The exorcism will probably involve thinking about how to come up with clean probabilistic models that (a) allow fast bottom-up inference and (b) don't involve tangling with likelihoods with as complex a form as those commonly used. We expect an exciting struggle with this problem over the next few years.

Space has forced us to omit some topics, important in radar tracking and likely to become important in vision. Firstly, the radar community is accustomed to generating tracks automatically; this practice is not unknown in vision, but most trackers are initialized by hand. Secondly, the radar community is accustomed to tracking multiple targets, with multiple returns; this complicates data association significantly, which is now seen as a weighted matching problem and dealt with using one of the many exact algorithms for that problem. Thirdly, the radar community is accustomed to dealing with tracking for objects that can switch dynamic model — this leads to so-called IMM filters, where a series of filters with distinct dynamic models propose different updates, and these proposals are weighted by evidence and updated. There is a little work on this topic in the vision community, but it tends to be somewhat difficult to do with a particle filter (actually, this means it needs an inconvenient number of particles; you can do pretty much anything with a particle filter, if you have enough particles).

There are other methods for maintaining approximations of densities. One might, for example, use a mixture of Gaussians with a constant number of components. It is rather natural to do data association by averaging, which will result in the number of elements in the mixture going up at each step; one is then supposed to cluster the elements and cull some components. We haven't seen this method used in vision circles yet.

Desiderata for a tracking application are:

- that tracks are initiated automatically;
- that tracks can be discarded automatically, as necessary (this means that the occasional erroneous track won't affect the count of total objects);
- that the tracker can be shown to work robustly over long sequences of data.

We discussed relatively few of the many kinematic human trackers, because they cannot meet these tests. It would be nice if this remark were obsolete by the time this book reaches its readers, but we don't think this will be the case (which is why we made it!). Tracking people on a general background remains extremely challenging; the difficulty is knowing how to initiate the track, which is hard because the variations in the appearance of clothing mean that it is generally difficult to know which pixels come from a person. Typically, current person trackers either initialize the tracker by hand, use aggressively simplified backgrounds which have high contrast with the moving person, or use background subtraction. These tricks are justified, because they make it possible to study this (extremely important) problem, but they yield rather unconvincing applications.

II Appendix: The Extended Kalman Filter, or EKF

We consider non-linear dynamic models of the form

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i); \Sigma_{d_i})$$

Again, we will need to represent with $P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$ (for prediction) and $P(\boldsymbol{x}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_i)$ (for correction). We take the position that these distributions can be represented by supplying a mean and a covariance. Typically, the representation works only for distributions that look rather like normal distributions — a big peak at one spot, and then a fast falloff. To obtain an extended Kalman filter, we linearize the dynamics about the current operating point, and linearize the measurement model. We do not derive the filter equations (it's a dull exercise in Laplace's approximation to integrals), but simply present them in algorithm **??**. We write the Jacobian of a function $\boldsymbol{g}$ — this is the matrix whose l, m'th entry is

$$\frac{\partial f_l}{\partial x_m}$$

— as $\mathcal{J}(\boldsymbol{g})$, and when we want to show that it has been evaluated at some point $\boldsymbol{x}_j$, we write $\mathcal{J}(\boldsymbol{g}; \boldsymbol{x}_j)$.

Dynamic Model:

$$\boldsymbol{x}_i \sim N(\boldsymbol{f}(\boldsymbol{x}_{i-1}, i), \Sigma_{d_i})$$

$$\boldsymbol{y}_i \sim N(\boldsymbol{h}(\boldsymbol{x}_i, i), \Sigma_{m_i})$$

Start Assumptions: $\overline{\boldsymbol{x}}_0^-$ and Σ_0^- are known

Update Equations: Prediction

$$\overline{\boldsymbol{x}}_i^- = \boldsymbol{f}(\overline{\boldsymbol{x}}_{i-1}^+)$$

$$\Sigma_i^- = \Sigma_{d_i} + \mathcal{J}(\boldsymbol{f}; \overline{\boldsymbol{x}}_{i-1}^+)^{-T} \Sigma_{i-1}^+ \mathcal{J}(\boldsymbol{f}; \overline{\boldsymbol{x}}_{i-1}^+)^{-1}$$

Update Equations: Correction

$$\mathcal{K}_i = \Sigma_i^- \mathcal{J}^T_{\boldsymbol{h};\overline{\boldsymbol{x}}_i^-} \left[\mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-) \Sigma_i^- \mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-)^T + \Sigma_{m_i}\right]^{-1}$$

$$\overline{\boldsymbol{x}}_i^+ = \overline{\boldsymbol{x}}_i^- + \mathcal{K}_i \left[\boldsymbol{y}_i - \boldsymbol{h}(\overline{\boldsymbol{x}}_i^-, i)\right]$$

$$\Sigma_i^+ = \left[Id - \mathcal{K}_i \mathcal{J}(\boldsymbol{h}; \overline{\boldsymbol{x}}_i^-)\right] \Sigma_i^-$$

Algorithm 18.10: *The extended Kalman filter maintains estimates of the mean and covariance of the various distributions encountered while tracking a state variable of some fixed dimension using the given non-linear dynamic model.*

Part VI

HIGH-LEVEL VISION

Chapter 19

CORRESPONDENCE AND POSE CONSISTENCY

This chapter poses object recognition as a correspondence problem — which image feature corresponds to which feature on which object? This simple view of recognition — which will naturally focus on the relationship between object features, image features and camera models — is useful.

We will discuss a variety of different algorithms that use this correspondence approach. The key observation underlying these algorithms is that objects do not scatter features in the image; if we know correspondences for a small set of features, it is fairly easy to obtain correspondences for a much larger set. This is because cameras are fairly orderly, and have relatively few degrees of freedom.

There are a number of practical reasons to understand the relationship between the position of image features, and the position and orientation of an object. In section 19.6, we describe one application, which uses the techniques from the rest of the chapter to register medical images with actual patients so that a surgeon can see where features in the image lie on the patient.

19.1 Initial Assumptions

All the algorithms we discuss assume that there is a collection of geometric models of the objects that should be recognised. This collection is usually referred to as the **modelbase**. We shall assume that, if information about an object turns out to be useful in an algorithm, we can ensure it is in the modelbase.

All the algorithms we describe in this chapter are of a single type, usually known as **hypothesize and test**. Each algorithm will:

- Hypothesize a correspondence between a collection of image features and a collection of object features, and then use this to generate a hypothesis about the projection from the object coordinate frame to the image frame. There are a variety of different ways of generating hypotheses. When camera intrinsic parameters are known, the hypothesis is equivalent to a hypothetical position and orientation — **pose** — for the object.

- Use this projection hypothesis to generate a rendering of the object. This step is usually known as **backprojection**[1].

- Compare the rendering to the image, and, if the two are sufficiently similar, accept the hypothesis.

For this approach to be effective, we must generate relatively few hypotheses, relatively quickly, and have good methods for comparing renderings and images. The process of comparison is usually called **verification** and can be quite unreliable; we describe verification techniques in section 19.5. Generally, these methods compute some score of the hypothesis that an object is present at a particular pose, which we shall call the **verification score**.

This approach works for point features and for curved surfaces, although the details are very much more difficult for curved surfaces. The vast majority of the literature deals with object models that consist of geometric features *that project like points*. This means that different views of the object give different views of the same set of features — though some may be occluded, etc. — rather than of different sets of features. We shall deal mainly with this case (which is by far the most useful in practice). However, in section 19.7, we describe some methods for obtaining and verifying hypotheses for images of curved surfaces.

We generally avoid detailed discussion of the question of what features should be matched. Most of the algorithms that we describe involve a certain amount of search amongst features — clearly, if we can describe features well, then this search is going to be reduced. For example, if our features are simply image points — perhaps obtained by intersecting edge curves — all points are equivalent and there may be a fair amount of search. If, instead, our points are described by the interest operators of section **??**, then the number of available correspondences goes down, and so does the amount of search required.

19.1.1 Obtaining Hypotheses

The main difference between algorithms is the mechanism by which hypotheses are obtained. The most obvious approach is to take all M geometric features in the image and all N geometric features on each of the L objects, and enumerate all the possible correspondences between object and image features — i.e. image feature 3 corresponds to feature 5 on object 7, etc. This is a terrible algorithm, because the number of possible correspondences is enormous — $O(LM^N)$.

Geometric constraints between object points limit the size of this space. For example, if we were matching 3D models to 3D data, we would expect pairs of points on the model to be the same distance apart as corresponding pairs of points on the data. Any correspondence for which this constraint is violated can be ignored, whatever the other components. This reasoning is equivalent to pruning a search tree — the approach of searching an aggressively pruned search tree is known as an **interpretation tree** algorithm, after work of Grimson and Lozano-Pérez [].

---

[1] For no reason we know; "projection", "forward projection" or "rendering" all seem more reasonable names.

Geometric constraints also apply when 3D models must be matched to 2D data. This is because the parameters of the projection model can usually be determined from a fairly small number of point correspondences. Once these parameters are known, the position of all other projected features is known too — this is a constraint, because we can't choose the position of these features arbitrarily. We can exploit these constraints by determining the projection parameters explicitly from a small number of correspondences, and then using the projection model to predict other correspondences (section 19.2 describes this well established strategy, which appears to originate with Hébert)

In fact, it isn't necessary to determine the projection parameters, by the following argument. Once we have established a correspondence between a small number of object features and a small number of image features — the base set — the camera constraints could be used to predict the position of other image features. This means that, in an appropriate sense, the position of the other image features is fixed *relative to the base set*. By an appropriate interpretation of this relative position, we can obtain measurements that are independent of the projection parameters, and use these to identify the object (section 19.4.1).

19.2 Obtaining Hypotheses by Pose Consistency

Assume we have an image of some object obtained using a camera model of known type, but with unknown parameters — for example, we might be viewing an object in a calibrated perspective camera with unknown extrinsic parameters with respect to the object frame. If we hypothesize a match between a sufficiently large group of image features and a sufficiently large group of object features, then we can recover the missing camera parameters from this hypothesis (and so render the rest of the object). Methods of this form (algorithm 1) are known as **pose consistency methods**. We will describe this family of methods — whose general form is shown in figure 1 — with a set of examples. This form of algorithm is increasingly being called **alignment**[2]; the term refers to the idea that the object is being aligned with its image.

We are really dealing with a family of methods here, because the details depend on what is known about the camera and whether the objects are two- or three-dimensional. We call a group that can be used to yield a camera hypothesis a **frame group** (there can be both object and image frame groups).

---

[2] The name was coined for a relatively recent version of the algorithm, which appears in the literature in many forms. It is only relatively recently that the similarity between these forms has become apparent.

```
For all object frame groups $O$
  For all image frame groups $F$
    For all correspondences $C$ between
      elements of $F$ and elements
      of $O$

      Use $F$, $C$ and $O$ to infer the missing parameters
      in a camera model

      Use the camera model estimate to render the object

      If the rendering conforms to the image,
        the object is present
    end
  end
end
```

Algorithm 19.1: *Alignment: matching object and image groups to infer a camera model*

19.2.1 Pose Consistency for Perspective Cameras

Assume we have a perspective camera for which the intrinsic parameters are known. This camera is viewing an object in the modelbase. Let us work in the object's coordinate system — the extrinsic parameters now boil down to the position and orientation of the camera in the object's frame. Now if we use algorithm 1, we have correspondences between a group of image features in an image coordinate system and a group of object features in the object coordinate system. From this information we can determine the extrinsic parameters of the camera (as in section **??**); but once the extrinsic parameters are known, the entire camera is known, and we can use this to render the rest of the object.

There is a variety of frame groups available for this problem. Typically, good frame groups contain "few" features of several different types (to reduce the number of correspondences to be searched). Groups that have been popular include:

- three points;
- three directions — often known as a **trihedral vertex** — and a point (which is necessary to establish scale);
- and a **dihedral vertex** (two directions emanating from a shared origin) and a point.

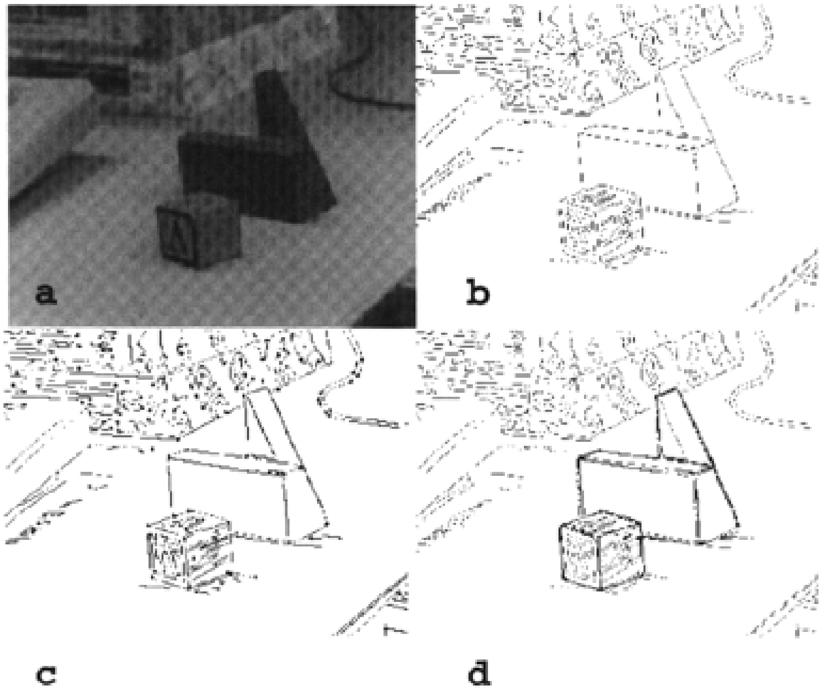


Figure 19.1. Pose consistency techniques work well for 3D objects, as long as one is using features that behave like points. This figure shows results from [?] (and is taken from page 50 of Ullman's book in the fervent hope that MIT press will give permission); three distinct polyhedral objects have been identified within an image. The top left image shows three objects on a table; at the top right we have edge points for that image; bottom left, the edge points chosen to form features (edge points with particular geometrical properties) and at the bottom right, the outlines of the polyhedra recognised superimposed the edge points. In this case, hypotheses were obtained by searching correspondences between pairs of image feature points and pairs of model points, and the camera model is affine uncalibrated.

Usually, directions are obtained by using line segments. This is attractive, because it is quite often the case that it is quite likely that part of a line segment will appear, but it is often difficult to localise the endpoints exactly.

The Intrinsic Parameters

It is quite common in the literature to assume that the intrinsic parameters of the camera are unknown, too. This doesn't change the problem all that much — although we might need to use more complicated frame groups — but it does offer

Missing
Figure

Figure 19.2. Reasoning about the intrinsic parameters of shared camera hypotheses can be used to disambiguate some perspective recognition problems.

some opportunities for more aggressive consistency reasoning. In images *with more than one object* we can require that the *intrinsic* camera parameters are the same for different objects.

The line of reasoning is quite simple. Firstly, we use algorithm 1 to recognise individual objects. Associated with each object is a camera solution. Now for each *pair* of recognised objects, we compare the intrinsic parameters of the camera solution: if they are (sufficiently) different, then the two hypotheses are incompatible. Figure 19.2 illustrates this simple trick, apparently due to [**?**].

19.2.2 Affine and Projective Camera Models

Calibrating perspective cameras is complicated, because the extrinsic parameters involve a rotation. It is often possible to use a camera model that allows simpler calibration, at the possible cost of greater ambiguity in the model identity. The two important simplifications are:

- **Affine cameras**, which model a perspective view as an affine transformation followed by an orthographic projection.
- **Projective cameras**, which model a perspective view as a projective transformation followed by perspective projection.

We will deal with each case in some detail; remember, the only real issue here is how to obtain a camera model from an hypothesized correspondence between an object frame group and an image frame group — the rest is supplied by algorithm 1.

Affine Cameras

In homogeneous coordinates we can write an affine camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and Π is an orthographic camera transformation. For reference, this means that

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\mathcal{A} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi\mathcal{A}\boldsymbol{P}_1$).

One possible frame group consists of four points. In this case, we must determine the camera — essentially, $\mathcal{A}$ — from a correspondence between four image points and four object points Now assume we have a (hypothesized) correspondence between four image points ($\boldsymbol{p}_i$) and four object points ($\boldsymbol{P}_i$). We can interpret $\boldsymbol{p}_i = \Pi\mathcal{A}\boldsymbol{P}_i$ as two linear equations in the first two rows of $\mathcal{A}$, that is

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \begin{pmatrix} a_{00}P_{i0} + a_{01}P_{i1} + a_{02}P_{i2} + a_{03} \\ a_{10}P_{i0} + a_{11}P_{i1} + a_{12}P_{i2} + a_{13} \end{pmatrix}$$

There are eight elements in the first two rows of $\mathcal{A}$, and with four points in general position we can solve these equations to obtain a unique solution for the first two rows of $\mathcal{A}$. Notice that the rest of $\mathcal{A}$ doesn't contribute to the projection, and doesn't need to be known to compute the projection of all other points. This means that knowing the first two rows of $\mathcal{A}$ is all we need to know to generate a backprojection.

Some models that are distinct under rotations and translations are ambiguous under affine cameras. Assume that one model is given by a set of points $\boldsymbol{P}_j$ and a second is given by $\boldsymbol{Q}_j$ *and* there is some affine transformation $\mathcal{B}$ such that for each j $\boldsymbol{P}_j = \mathcal{B}\boldsymbol{Q}_j$. These models can't be distinguished under an affine camera. A view of the first model in an affine camera is a set of image points $\boldsymbol{p}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j$ and a view of the second model in some other affine camera is given by a set of image points $\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j$. Now if $\mathcal{A}_2 = \mathcal{A}_1\mathcal{B}$, then we have that

$$\boldsymbol{q}_j = \Pi\mathcal{A}_2\boldsymbol{Q}_j = \Pi\mathcal{A}_1\mathcal{B}\boldsymbol{Q}_j = \Pi\mathcal{A}_1\boldsymbol{P}_j = \boldsymbol{p}_j$$

that is, that there is an affine camera that makes the second model look exactly like a view of the first model in some other affine camera — so they are indistinguishable.

Projective Cameras

In homogeneous coordinates we can write a projective camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general projective transformation and Π is a perspective camera transformation. For reference, this means that

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

and

$$\mathcal{A} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Again, we use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi \mathcal{A} \boldsymbol{P}_1$). Notice that, because we are working in homogenous coordinates, $\mathcal{A}$ and $\lambda \mathcal{A}$ represent the same transformation if $\lambda \neq 0$.

One possible frame group consists of five points. In this case, we must determine the camera — essentially, $\mathcal{A}$ — from a correspondence between five image points and five object points Now assume we have a (hypothesized) correspondence between five image points ($\boldsymbol{p}_i$) and five object points ($\boldsymbol{P}_i$). We can interpret $\boldsymbol{p}_i = \Pi \mathcal{A} \boldsymbol{P}_i$ as two *non-linear* equations in the first two rows of $\mathcal{A}$, that is

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \frac{1}{a_{30}P_{i0} + a_{31}P_{i1} + a_{32}P_{i2} + a_{33}} \begin{pmatrix} a_{00}P_{i0} + a_{01}P_{i1} + a_{02}P_{i2} + a_{03} \\ a_{10}P_{i0} + a_{11}P_{i1} + a_{12}P_{i2} + a_{13} \end{pmatrix}$$

There are twelve elements in the first three rows of $\mathcal{A}$, and with five points in general position we can solve these equations to obtain a unique solution for the first three rows of $\mathcal{A}$. Notice that the rest of $\mathcal{A}$ doesn't contribute to the projection, and doesn't need to be known to compute the projection of all other points. This means that knowing the first three rows of $\mathcal{A}$ is all we need to know to generate a backprojection. Notice also that all we have done here is to repeat — in significantly less detail — the activities of section 19.2.2.

Some models that are distinct under affine cameras (and so under rotations and translations) are ambiguous under projective cameras. Assume that one model is given by a set of points $\boldsymbol{P}_j$ and a second is given by $\boldsymbol{Q}_j$ *and* there is some projective transformation $\mathcal{B}$ such that for each j $\boldsymbol{P}_j = \mathcal{B}\boldsymbol{Q}_j$. These models can't be distinguished under a projective camera. A view of the first model in a projective camera is a set of image points $\boldsymbol{p}_j = \Pi \mathcal{A}_1 \boldsymbol{P}_j$ and a view of the second model in some other projective camera is given by a set of image points $\boldsymbol{q}_j = \Pi \mathcal{A}_2 \boldsymbol{Q}_j$. Now if $\mathcal{A}_2 = \mathcal{A}_1 \mathcal{B}$, then we have that

$$\boldsymbol{q}_j = \Pi \mathcal{A}_2 \boldsymbol{Q}_j = \Pi \mathcal{A}_1 \mathcal{B} \boldsymbol{Q}_j = \Pi \mathcal{A}_1 \boldsymbol{P}_j = \boldsymbol{p}_j$$

that is, that there is a projective camera that makes the second model look exactly like a view of the first model in some other projective camera — so they are indistinguishable.

19.2.3 Linear Combinations of Models

The case of an affine camera used the correspondences to perform explicit camera calibration. We can hide the camera calibration process with a little linear algebra. We use homogeneous coordinates, and can write a general uncalibrated affine camera as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We use capital letters for points on the model, small letters for points in the image, and subscripts to denote correspondence (so that $\boldsymbol{p}_1 = \Pi\mathcal{A}\boldsymbol{P}_1$).

Let us identify one point on the model as an origin, and consider offsets from that point (this means that translations can be ignored). Use the notation $\boldsymbol{v}_i = \boldsymbol{p}_i - \boldsymbol{p}_0$ and $\boldsymbol{V}_i = \boldsymbol{P}_i - \boldsymbol{P}_0$. Now obtain three views of the object in different general affine cameras — with affine transformations $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, so that for the i'th point on the object we have

$$\begin{aligned} \boldsymbol{v}_i^A &= \Pi\mathcal{A}\boldsymbol{V}_i \\ \boldsymbol{v}_i^B &= \Pi\mathcal{B}\boldsymbol{V}_i \\ \boldsymbol{v}_i^C &= \Pi\mathcal{C}\boldsymbol{V}_i \end{aligned}$$

Because Π contains a lot of zeros, and the fourth row of $\boldsymbol{V}_i$ is zero, we can simplify things considerably with these three views.

Write the j'th **row** of $\mathcal{A}$ as ${\boldsymbol{a}^T}_j$. We now have

$$\begin{aligned} \boldsymbol{v}_i^A &= (\boldsymbol{a}_0^T.\boldsymbol{V}_i, \boldsymbol{a}_1^T.\boldsymbol{V}_i, 0)^T \\ \boldsymbol{v}_i^B &= (\boldsymbol{b}_0^T.\boldsymbol{V}_i, \boldsymbol{b}_1^T.\boldsymbol{V}_i, 0)^T \\ \boldsymbol{v}_i^C &= (\boldsymbol{c}_0^T.\boldsymbol{V}_i, \boldsymbol{c}_1^T.\boldsymbol{V}_i, 0)^T \end{aligned}$$

We would now like to generate some arbitrary new view of the object, which could be obtained by applying $\Pi\mathcal{D}$ to the points, where $\mathcal{D}$ is some new affine transformation. To obtain this view, we must first decide where $\boldsymbol{p}_0$ lies; having done so, we need the $\boldsymbol{v}_i^D$ for the i'th point.

Now $\boldsymbol{v}_i^D = (\boldsymbol{d}_0^T.\boldsymbol{V}_i, \boldsymbol{d}_1^T.\boldsymbol{V}_i, 0)$. Assuming that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are general, we have that $\boldsymbol{d}_j$ must be a fixed linear combination of $\boldsymbol{a}_j$, $\boldsymbol{b}_j$ and $\boldsymbol{c}_j$, say

$$\boldsymbol{d}_j = \lambda_{(\boldsymbol{a}_j)}\boldsymbol{a}_j + \lambda_{(\boldsymbol{b}_j)}\boldsymbol{b}_j + \lambda_{(\boldsymbol{c}_j)}\boldsymbol{c}_j$$

Then we have that

$$\boldsymbol{v}_i^D = (\lambda_{(\boldsymbol{a}_0)}\boldsymbol{a}_0^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{b}_0)}\boldsymbol{b}_0^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{c}_0)}\boldsymbol{c}_0^T.\boldsymbol{V}_i, \lambda_{(\boldsymbol{a}_1)}\boldsymbol{a}_1^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{b}_1)}\boldsymbol{b}_1^T.\boldsymbol{V}_i + \lambda_{(\boldsymbol{c}_1)}\boldsymbol{c}_1^T.\boldsymbol{V}_i, 0)$$

which means that, given three unknown affine views of the object, we can reconstruct a fourth by determining the values of these λ's.

This strategy has become known as **linear combinations of models**. Generating hypotheses with this method requires searching correspondences, too; we select some image points to be $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, etc. and then solve for the λ values. Once these are known, we can render the object, although additional ingenuity is required for hidden line removal. Notice that the approach is simply an alternative version of affine camera calibration. It has the attractive feature that the object model is constructed from three views of the object in a fairly simple way. It turns out that an object model is also easily constructed from three views for the approach of section 19.4.1, too.

19.3 Obtaining Hypotheses by Pose Clustering

Most objects have many frame groups. This means that there should be many correspondences between object and image frame groups that will verify satisfactorily. Each of these correspondences should yield approximately the same estimate of position and orientation for the object with respect to the camera (or the camera with respect to the object — it doesn't matter which we work with). However, image frame groups that come from noise (or **clutter**, a term used for objects that are not of interest and not in the modelbase) are likely to yield estimates of pose that are uncorrelated. This motivates the use of some form of clustering method to filter hypotheses before verification.

For each object, we set up an accumulator array that represents pose space — each element in the accumulator array corresponds to a "bucket" in pose space. Now we take each image frame group, and hypothesize a correspondence between it and every frame group on every object. For each of these correspondences, we determine pose parameters and make an entry in the accumulator array for the current object at the pose value. If there are large numbers of votes in any object's accumulator array, this can be interpreted as evidence for the presence of that object at that pose; this evidence can be checked using a verification method. It is important to note the similarity between this method (which is given in algorithm 2) and the Hough transform (section **??**).

There are two difficulties with these methods (which mirror the difficulties in using the Hough transform in practice).

1. In an image containing noise or texture that generates many spurious frame groups, the number of votes in the pose arrays corresponding to real objects may be smaller than the number of spurious votes (the details are in []).

2. Choosing the size of the buckets in the pose arrays is difficult; buckets that are too small mean that there is no accumulation of votes (because it is hard to compute pose accurately); buckets that are too large mean that too many buckets will have enough votes to trigger a verification attempt.

We can improve the noise resistance of the method by not counting votes for objects at poses where the vote is obviously unreliable — for example, in cases where, if the object was at that pose, the object frame group would be invisible. These improvements are sufficient to yield working systems (figure 19.3).

```
For all objects $O$
  For all object frame groups $F(O)$
    For all image frame groups $F(I)$
      For all correspondences $C$ between
        elements of $F(I)$ and elements
        of $F(O)$

        Use $F(I)$, $F(O)$ and $C$ to infer object pose $P(O)$

        Add a vote to $O$'s pose space at the bucket
        corresponding to $P(O)$.
      end
    end
  end
end
For all objects $O$
  For all elements $P(O)$ of $O$'s pose space that have
    enough votes

    Use the $P(O)$ and the
    camera model estimate to render the object

    If the rendering conforms to the image,
    the object is present
  end
end
```

Algorithm 19.2: *Pose clustering: voting on pose, correspondence and identity*

19.4 Obtaining Hypotheses Using Invariants

Pose clustering methods collect correspondences that imply similar hypotheses about camera calibration and pose. Another way to obtain object hypotheses is to use measurements that are independent of the camera properties. This approach is most easily developed for images of planar objects, but can be applied to other cases as well (section **??**).

19.4.1 Invariants for Plane Figures

Recall that an affine camera can be written as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general affine transformation and Π is a orthographic camera transformation. Assume we have

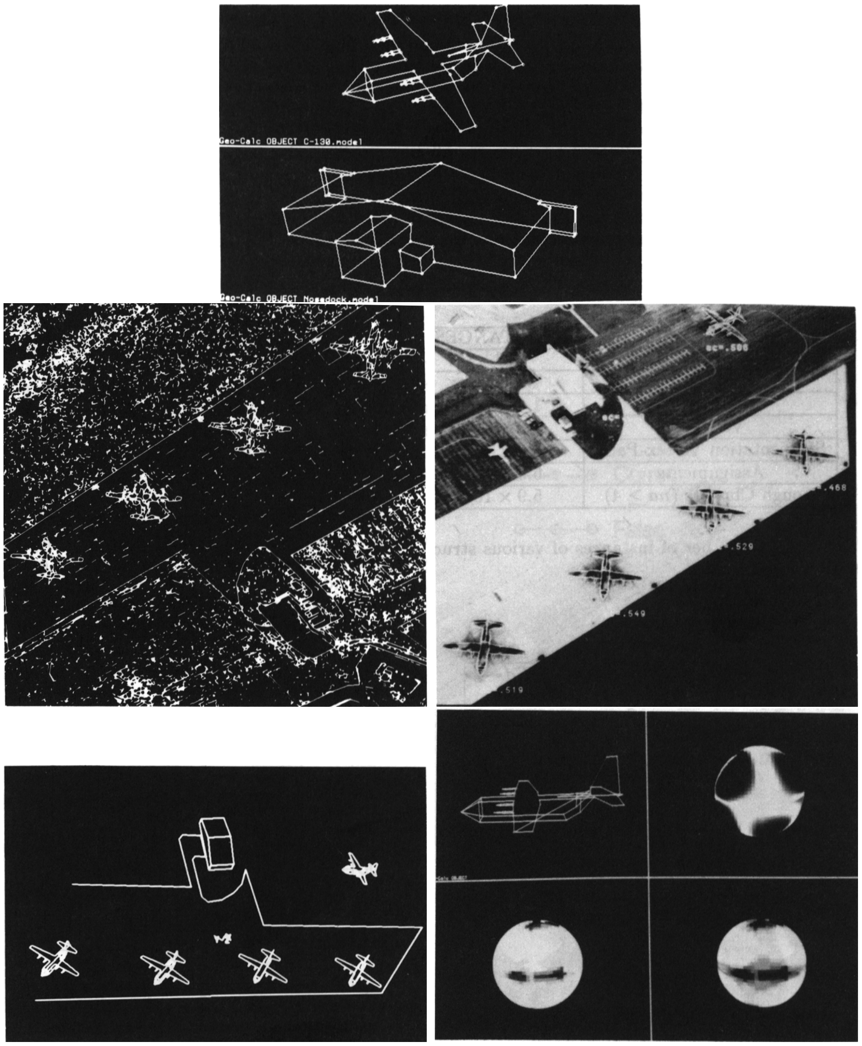


Figure 19.3. Pose clustering methods use frame-bearing groups to generate pose estimates, and then cluster these estimates. **Top:** two models used in an early pose clustering system. **Center left:** edge points marked for an image used in testing. **Center right:** edges of models that are found, overlaid on the image. **Bottom left:** a new view of the layout of the models in space, to indicate their pose; notice the curious pose of the aircraft off the runway. **Bottom right:** for each framebearing group, some views are better than others because the estimate of pose will be more stable; next to the model of the aircraft, we see a sphere representing different viewpoints, with light regions corresponding to high error views of the pair of features marked on the model

a set of model points $\boldsymbol{P}_j$, which are coplanar; without loss of generality, we can assume they lie on the $z = 0$ plane. Now we have

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \Pi \mathcal{A} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 0 \\ 1 \end{pmatrix}$$

(using the notation of section **??**). We can substitute for Π and $\mathcal{A}$ to obtain

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 1 \end{pmatrix}$$

This is important; it means that views of a set of coplanar points in an affine camera are generated by **plane affine transformations** — this means that we can abstract away the camera, and reason only about the effect of these transformations on the model.

A similar result applies to views of plane points in a projective camera — in this case, the transformation is a **plane projective transformation**. To get this result, recall that a projective camera can be written as $\Pi\mathcal{A}$, where $\mathcal{A}$ is a general projective transformation and Π is a perspective camera transformation. Assume we have a set of model points $\boldsymbol{P}_j$, which are coplanar; without loss of generality, we can assume they lie on the $z = 0$ plane. Now we have

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ 1 \end{pmatrix} = \Pi \mathcal{A} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 0 \\ 1 \end{pmatrix}$$

(using the notation of section 19.2.2). We can substitute for Π and $\mathcal{A}$ to obtain

$$\begin{pmatrix} p_{i0} \\ p_{i1} \end{pmatrix} = \frac{1}{a_{20}P_{i0} + a_{21}P_{i1} + a_{23}} \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ 1 \end{pmatrix}$$

Recalling that we are working in homogenous coordinates, a more convenient form is:

$$\begin{pmatrix} p_{i0} \\ p_{i1} \\ p_{i2} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{03} \\ a_{10} & a_{11} & a_{13} \\ a_{20} & a_{21} & a_{23} \end{pmatrix} \begin{pmatrix} P_{i0} \\ P_{i1} \\ P_{i3} \end{pmatrix}$$

Again, this means that views of a set of coplanar points in a projective camera are generated by these plane projective transformations — this means that we can abstract away the camera, and reason only about the effect of these transformations on the model.

Affine Invariants for Co-planar Points

Assume we have a model which is a set of coplanar points. Choose three of these points $\boldsymbol{P}_0$, $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. This gives a coordinate frame, and any other point $\boldsymbol{P}_i$ in the model can be expressed as $\boldsymbol{P}_0 + \mu_{i1}(\boldsymbol{P}_1 - \boldsymbol{P}_0) + \mu_{i2}(\boldsymbol{P}_2 - \boldsymbol{P}_0)$ — it takes only a little linear algebra to compute the μ values associated with each points.

Now the camera takes model points $\boldsymbol{P}_i$ to image points $\boldsymbol{p}_i$. For an uncalibrated affine camera viewing a set of plane points, the effect of the camera can be written as an (unknown) plane affine transformation. We can write the camera as $\mathcal{C}$. Now

$$\begin{aligned}
\boldsymbol{p}_i &= \mathcal{C}\boldsymbol{P}_i \\
&= \mathcal{C}(\boldsymbol{P}_0 + \mu_{i1}(\boldsymbol{P}_1 - \boldsymbol{P}_0) + \mu_{i2}(\boldsymbol{P}_2 - \boldsymbol{P}_0)) \\
&= (1 - \mu_{i1} - \mu_{i2})(\mathcal{C}\boldsymbol{P}_0) + \mu_{i1}(\mathcal{C}\boldsymbol{P}_1) + \mu_{i2}(\mathcal{C}\boldsymbol{P}_2) \\
&= (1 - \mu_{i1} - \mu_{i2})\boldsymbol{p}_0 + \mu_{i1}\boldsymbol{p}_1 + \mu_{i2}\boldsymbol{p}_2 \\
&= \boldsymbol{p}_0 + \mu_{i1}(\boldsymbol{p}_1 - \boldsymbol{p}_0) + \mu_{i2}(\boldsymbol{p}_2 - \boldsymbol{p}_1)
\end{aligned}$$

This means that the μ_{ij} describe the geometry of the object, and *are independent of the view* — i.e. if we compute the μ_{ij} in the model plane or in some affine view, we will obtain the same values. Measurements with this property are often referred to as **affine invariants** (other constructions for affine invariants are given in the exercises).

Projective Invariants for Co-planar Points and Lines

In homogenous coordinates, we can write the relationship between image points and (plane) model points as $\boldsymbol{p}_i = \mathcal{A}\boldsymbol{P}_i$, where $\mathcal{A}$ is a general 3x3 matrix. Now we have that

$$\begin{aligned}
\frac{\det[\boldsymbol{p}_i\boldsymbol{p}_j\boldsymbol{p}_k]}{\det[\boldsymbol{p}_i\boldsymbol{p}_l\boldsymbol{p}_m]}\frac{\det[\boldsymbol{p}_i\boldsymbol{p}_j\boldsymbol{p}_l]}{\det[\boldsymbol{p}_i\boldsymbol{p}_k\boldsymbol{p}_m]} &= \frac{\det[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_j)(\mathcal{A}\boldsymbol{P}_k)]}{\det[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_l)(\mathcal{A}\boldsymbol{P}_m)]}\frac{\det[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_j)(\mathcal{A}\boldsymbol{P}_l)]}{\det[(\mathcal{A}\boldsymbol{P}_i)(\mathcal{A}\boldsymbol{P}_k)(\mathcal{A}\boldsymbol{P}_m)]} \\
&= \frac{\det\mathcal{A}\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_k]}{\det\mathcal{A}\det[\boldsymbol{P}_i\boldsymbol{P}_l\boldsymbol{P}_m]}\frac{\det\mathcal{A}\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_l]}{\det\mathcal{A}\det[\boldsymbol{P}_i\boldsymbol{P}_k\boldsymbol{P}_m]} \\
&= \frac{\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_k]}{\det[\boldsymbol{P}_i\boldsymbol{P}_l\boldsymbol{P}_m]}\frac{\det[\boldsymbol{P}_i\boldsymbol{P}_j\boldsymbol{P}_l]}{\det[\boldsymbol{P}_i\boldsymbol{P}_k\boldsymbol{P}_m]}
\end{aligned}$$

(as long as no two of i, j, k, l, and m are the same and no three of the points are collinear). There are other arrangements of determinants that are invariant as well (see the exercises).

Plane Algebraic Curves and Projective Transformations

Algebraic curves consist of all points on the plane where a polynomial vanishes. A line is an algebraic curve. If we write points in homogenous coordinates as $\boldsymbol{p}_i = [p_{i0}, p_{i1}, p_{i2}]^T$, a line is is the locus of points $\boldsymbol{p}$ for which $l_0p_0 + l_1p_1 + l_2p_2 = 0$

— we can write this as $\boldsymbol{l}^T\boldsymbol{p} = 0$, where the line is represented by $\boldsymbol{l}$. Now if our points transform by $\boldsymbol{p} = \mathcal{A}\boldsymbol{P}$, then the lines will transform by $\boldsymbol{l} = \mathcal{A}^{-T}\boldsymbol{L}$. This is easiest to see by observing that

$$\boldsymbol{l}^T\boldsymbol{p} = \boldsymbol{l}^T\mathcal{A}\boldsymbol{P} = \boldsymbol{L}^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}$$

so that if $\boldsymbol{p}$ lies on line $\boldsymbol{l}$, then $\boldsymbol{P}$ lies on line $\boldsymbol{L}$.

Because lines transform (basically) like points, we have

$$\begin{aligned}
\frac{\det[\boldsymbol{l}_i\boldsymbol{l}_j\boldsymbol{l}_k]}{\det[\boldsymbol{l}_i\boldsymbol{l}_l\boldsymbol{l}_m]}\frac{\det[\boldsymbol{l}_i\boldsymbol{l}_j\boldsymbol{l}_l]}{\det[\boldsymbol{l}_i\boldsymbol{l}_k\boldsymbol{l}_m]} &= \frac{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_j)(\mathcal{A}^{-T}\boldsymbol{L}_k)\right]}{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_l)(\mathcal{A}^{-T}\boldsymbol{L}_m)\right]}\frac{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_j)(\mathcal{A}^{-T}\boldsymbol{L}_l)\right]}{\det\left[(\mathcal{A}^{-T}\boldsymbol{L}_i)(\mathcal{A}^{-T}\boldsymbol{L}_k)(\mathcal{A}^{-T}\boldsymbol{L}_m)\right]} \\
&= \frac{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_k]}{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_l\boldsymbol{L}_m]}\frac{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_l]}{\det\mathcal{A}\det[\boldsymbol{L}_i\boldsymbol{L}_k\boldsymbol{L}_m]} \\
&= \frac{\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_k]}{\det[\boldsymbol{L}_i\boldsymbol{L}_l\boldsymbol{L}_m]}\frac{\det[\boldsymbol{L}_i\boldsymbol{L}_j\boldsymbol{L}_l]}{\det[\boldsymbol{L}_i\boldsymbol{L}_k\boldsymbol{L}_m]}
\end{aligned}$$

(as long as no two of i, j, k, l, and m are the same and no three of the lines pass through a single point).

In fact, algebraic invariants abound in the projective case. Useful examples occur in particular for plane conics. A plane conic is the locus of points $\boldsymbol{x}$ such that $\boldsymbol{x}^t\mathcal{M}\boldsymbol{x} = 0$, where $\boldsymbol{x}$ is the vector of homogenous coordinates describing a point, and the matrix $\mathcal{M}$ contains the coefficients of the conic. Now if we transform the coordinate system somehow (say, by observing the points in a camera), then we have $\boldsymbol{x}' = \mathcal{P}\boldsymbol{x}$ for some plane projective transformation $\mathcal{P}$. The equation of the conic in the new coordinate system can be obtained by noticing that the new equation must vanish for every point that used to lie on the old conic — that is, for every point for which the old conic's equation vanished in the old coordinate frame. In particular, if we invert the transformation and plug the resulting point into the old conic's equation, we should get a zero. This line of reasoning means that $\mathcal{M}' = \mathcal{P}^{-t}\mathcal{M}\mathcal{P}^{-1}$ is the equation of the conic in the new frame.

Now assume that we have two conics, $\mathcal{M}$ and $\mathcal{N}$. Each transforms in this fashion, meaning that $\mathcal{A}_{MN} = \mathcal{M}^{-1}\mathcal{N}$ transforms to $\mathcal{A}'_{MN} = \mathcal{P}\mathcal{M}^{-1}\mathcal{N}\mathcal{P}^{-1}$, which we observe. This means, in turn, that the eigenvalues of $\mathcal{A}'_{MN}$ are the same as $\mathcal{A}_{MN}$. We can observe both $\mathcal{A}_{MN}$ —by looking at the "model" — and $\mathcal{A}'_{MN}$ — by looking at the image— but only up to a constant scale factor. This means that the eigenvalues we observe may have been scaled by a constant but unknown factor; however, appropriate ratios of eigenvalues will be invariant. A useful example is $trace(\mathcal{A}_{MN})^3/det(\mathcal{A}_{MN})$.

It is quite easy to construct invariants for mixed sets of points and lines, too. For example, assume we have a set of points $\boldsymbol{p}_i$ and a set of lines $\boldsymbol{l}_j$. Notice that

$$\begin{aligned}
\frac{\boldsymbol{l}_i^T\boldsymbol{p}_k\boldsymbol{l}_j\boldsymbol{p}_l}{\boldsymbol{l}_i^T\boldsymbol{p}_l\boldsymbol{l}_j\boldsymbol{p}_k} &= \frac{\boldsymbol{L}_i^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_k\boldsymbol{L}_j^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_l}{\boldsymbol{L}_i^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_l\boldsymbol{L}_j^T\mathcal{A}^{-1}\mathcal{A}\boldsymbol{P}_k} \\
&= \frac{\boldsymbol{L}_i^T\boldsymbol{P}_k\boldsymbol{L}_j^T\boldsymbol{P}_l}{\boldsymbol{L}_i^T\boldsymbol{P}_l\boldsymbol{L}_j^T\boldsymbol{P}_k}
\end{aligned}$$

which means that this expression is invariant, too (as long as i and j are not equal and k and l are not equal).

Projective invariants for various mixtures of points, lines and conics are known, and have been used successfully in object recognition (some examples are explored in exercises **??**-**??**). Projective invariants are known for plane algebraic curves of higher degree, but are of little practical significance because such curves are seldom encountered in practice and are hard to fit accurately.

19.4.2 Geometric Hashing

Geometric hashing is an algorithm that uses geometric invariants to vote for object hypotheses. You should keep pose clustering in mind as an analogy — we will be voting again — but we are now voting on geometry rather than on pose. The idea was originally developed for uncalibrated affine views of plane models, and is easiest to explain in this context.

For any set of three points on the model, we can use the techniques of section 19.4.1 to compute values of μ_1 and μ_2 for every other point in the model. We now set up a table indexed by the values of μ_1 and μ_2. For every model in the modelbase and for every group of three points on that model, we compute the μ's for every other point. Using these μ's as an index, insert an entry recording the name of the model and the three points on the model that gave rise to the values obtained. Thus, a pair of μ's acts as an hypothesis about the identity of the model *and* three points on that model.

Now we have the table, we can find the model by searching correspondences. We take any triple of *image* points, and compute the μ's for every other point in the image. We recover the contents of the table indexed by all of these μ's. If the triple corresponds to a triple on the object, then we are going to obtain many votes for the combination of the object *and* the three points. Hopefully, noise votes will be uncorrelated, meaning that there will be a lot of uncoordinated votes for various triples on various objects, and many votes for an object triple combination. This implies that, if we have many votes for the same object *and* the same three points on that object, the object may well be present. Notice that this set of three points can act as a frame group for verification purposes. Voting on the μ values is sketched in figure 3.

This algorithm can be generalised to work for other geometric groups than points (see the assignments!). If we have uncalibrated affine views of 3D objects, then there are three μ's for each point, and we cannot determine them uniquely for each point, but the method extends (see the assignments, again!). As with pose clustering and the hough transform (which is really what this method is), it is difficult to choose the size of the buckets; it is hard to be sure what "enough" means; and there is some danger that the table will get clogged.

```
For all groups of three image points $T(I)$
  For every other image point $p$

    Compute the $\mu$'s from $p$ and $T(I)$

    Obtain the table entry at these values
    if there is one, it will label the three points in $T(I)$
    with the name of the object
    and the names of these particular points.

    Cluster these labels;
    if there are enough labels, backproject and verify

    end
  end
end
```

Algorithm 19.3: *Geometric hashing: voting on identity and point labels*

19.4.3 Invariants and Indexing

Geometric hashing searches correspondences, but does so extracting acceptable labels from a hash table. The main feature of geometric hashing is that we do not need to search over models at recognition time — the hash table has been preloaded in a way that avoids this. This is a desirable feature, usually called **indexing**. One version of indexing applies in alignment when different models have different types of frame group; clearly, an image frame group of a particular kind need only be checked against the models for which that type of frame group applies.

The trick in geometric hashing is to look for image groups that contained information that was independent of the object pose and changed from object to object (the μ's). These μ's then generate object information. Geometric hashing explores all possible groups of points. The motivating trick can be extended to all sorts of other geometric features as well. We shall call groups of features that carry information that is independent of object pose and changes from object to object **invariant bearing groups** — we have seen some examples in section 19.4.1. Assume that we know the different types of invariant-bearing group that are available. We can now modify the alignment algorithm to come up with the algorithm of figure 4.

This approach could be extremely efficient if invariant bearing groups were distinctive, in the sense that there are very few model feature groups with the same values of the invariants. We also need to be able to measure the values of the invariants accurately. Notice that we have to look at every model feature group with

```
For each type $T$ of invariant-bearing group
  For each image group $G$ of type $T$

  Determine the values $V$ of the invariants of $G$

    For each model feature group $M$ of type $T$ whose invariants
    have the values $V$

      Determine the transformation that takes $M$ to $G$

      Render the model using this transformation

      Compare the result with the image, and accept if
      similar
    end
  end
end
```

Algorithm 19.4: *Invariant indexing using invariant bearing groups*

the same values as the image feature group, because we don't know which group we might have. This is, again, a search over correspondences, with a reasonable hope that the correspondences to be searched could be few.

Indexing in Uncalibrated Perspective Views with Lines and Conics

There are numerous invariant bearing groups that can be plugged into the general algorithm given above. The process works best with plane objects viewed in unknown perspective cameras, where invariants of the imaging transformations are quite freely available. Generally, the most useful cases appear to be the invariants of three kinds of groups: five lines; two conics; and a conic and two lines.

Given these functions, a typical system works like this:

- **Extract primitive groups:** Images are passed through an edge detector, and conics and lines are fitted to groups of edge points. The edge points are discarded, and the fitted curves retained. It is excessively onerous to look at all groups of five line segments to form invariants. It is also unnecessary, because objects do not consist of scattered lines, so that open curves of line segments are all that is required. These are groups where, at at least one end point of each line segment, there is an end point of another segment nearby.
- **Index using invariants:** Relevant assemblies of lines and conics are used to

obtain object hypotheses, usually by indexing in arrays using quantised values of the invariants. Typically there are one or two invariants available for each type of group, and relatively small numbers of models, so that using an array is not particularly wasteful. The size of the quantisation buckets is usually determined by trial and error; a wise implementer searches the neighbours of a selected bucket, too. Again, the fact that there are only one or two invariants for each type of group means that this is not too wasteful.

- **Back-project and verify:** For each object hypothesis, the transformation that takes the model group to the image group is determined. This transformation is used to backproject the model, and verification proceeds.

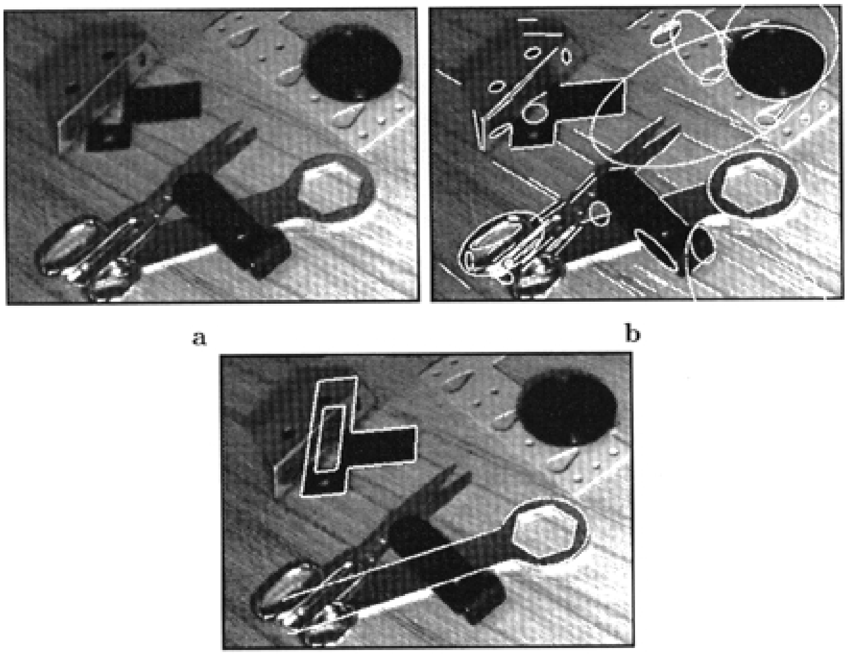


Figure 19.4. These figures illustrate the overall structure of a system that recognises plane objects using invariants. At the top left is an image; top right shows edge points that have been fitted with lines or conics; and at the bottom, outline points from the objects that have been recognised and verified have been overlaid (reproduced from [**?**], p. *** - as OUP will ideally give permission).

Invariant Indexing for Plane Curves

One source of geometric invariants is to use **covariant constructions** — constructions that commute with the transformation in mind, meaning that if you perform the construction and transform the result, you get the same result as if you transform the geometry and then perform the construction. In this approach, the construction yields a coordinate frame that is then transformed into some convenient universal coordinate frame, and measurements are made in that frame (usually called a **canonical frame**). Since the measurements are made in a fixed coordinate frame, any property measured in a canonical frame is an invariant. You should notice the similarity of this idea to that used in geometric hashing — the μ's there are measured in a canonical frame.

For example, take a curve and construct a line that is tangent to it at two distinct points (closed plane curves that are not convex have such lines — convex closed curves do not, and for open curves there are no guarantees). Now apply a transformation so that one of the points of tangency lies at the origin and the tangent line lies on the x-axis. In this coordinate system any measurement you care to take is an invariant, *if* you are careful about the fact that you may not know which point to place at the origin. This freedom brings with it a troubling question, which we put off addressing until section **??**: given we can make many different invariant measurements, which should we make?

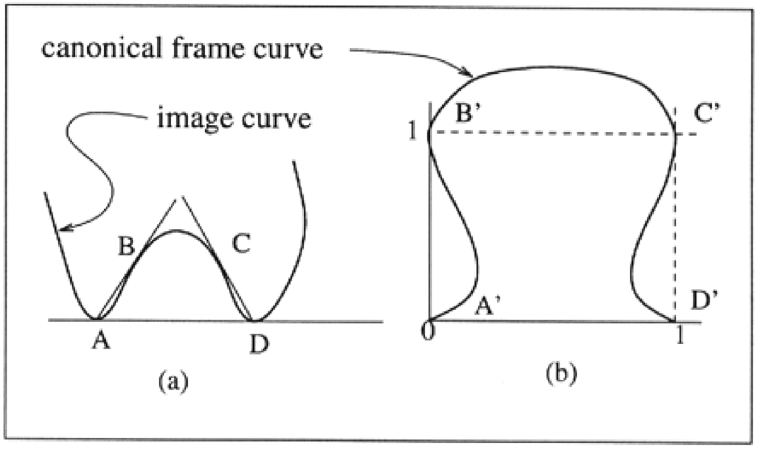


Figure 19.5. Because tangency and incidence are covariant (i.e. a tangent in one coordinate system will, when transformed, be a tangent in the other coordinate system), constructions based around tangency and incidence yield canonical coordinate frames. This figure (reproduced from [?], p. 123 - as OUP will ideally give permission) shows a construction called the *M-curve construction.* For a curve shaped like the letter M (upside-down, by convention!), a bitangent yields two points (A and D) from which tangents can be produced to meet the curve at B and C; this yields a total of four points. Since any set of four points with no three collinear can be mapped to any other such set, we can take these points to the unit square on the plane, and look at the curve in this coordinate system the *canonical frame.* Any measurement in this system will be invariant.

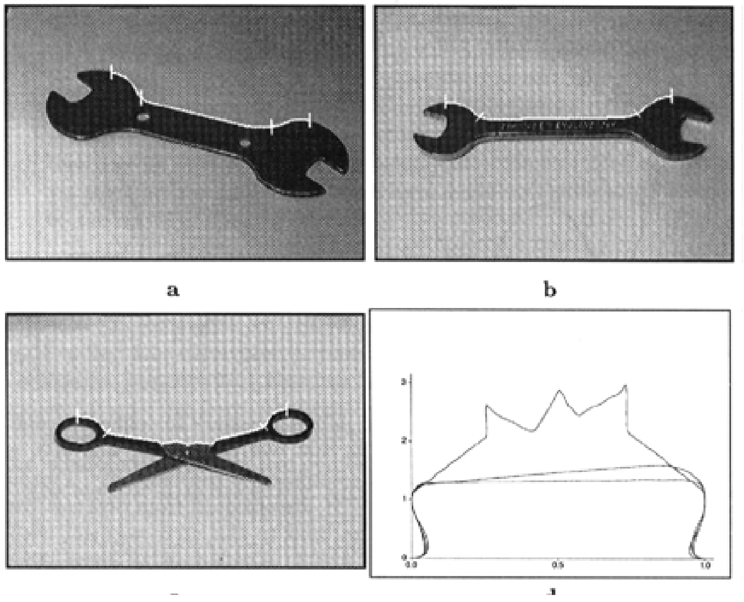


Figure 19.6. Canonical frames are interesting only if objects do look different in these frames. This figure (reproduced from [?], p. 124 - as OUP will ideally give permission) shows M-curves obtained from three different objects, viewed in isolation, and then the M-curves superimposed on the canonical frame. These curves look quite different, suggesting that the construction can yield useful measurements. Of course, this is not guaranteed; if one object is simply a scaled version of another, their M-curves are going to look the same in the canonical frame.

19.5 Verification

Accurate verification requires good tests for whether a rendering of an object model is similar to an image. The choice of test depends on the amount of information about the world available to generate the rendering. For example, if the lighting in the world and the camera response to illumination are both known precisely — for a system that viewed parts on a conveyor belt this might be possible — then we could reasonably expect the rendering to predict image pixel values accurately. In this case, comparing pixel values would be a sensible test.

Usually, all we know about the illumination is that it is bright enough to have generated an hypothesis. This means that comparisons should be robust to changes in illumination. The only test used in practice is to render the silhouette of the object, and then compare it to edge points in an image. We describe some other possible tests, too.

19.5.1 Edge Proximity

A natural test is to overlay object silhouette edges on the image, using the camera model, and then score the hypothesis by comparing these points with actual image edge points. The usual score is the fraction of the length of predicted silhouette edges that lie nearby actual image edge points. This is invariant to rotation and translation in the camera frame, which is a good thing, but changes with scale, which may not be a bad thing. It is usual to allow edge points to contribute to a verification score only if their orientation is similar to the orientation of the silhouette edge to which they are being compared. The principle here is that the more detailed the description of the edge point, the more likely one is to know whether it came from the object.

It is a bad idea to include invisible silhouette components in the score, so the rendering should be capable of removing hidden lines. The silhouette is used because edges internal to a silhouette may have low contrast under a bad choice of illumination. This means that their absence may be evidence about the illumination rather than the presence or absence of the object.

Edge proximity tests can be quite unreliable. Even orientation information doesn't really overcome these difficulties. When we project a set of model boundaries into an image, the *absence* of edges lying near these boundaries could well be a quite reliable sign that the model isn't there, but the *presence* of edges lying near the boundaries is *not* a particularly reliable sign the object is there. This is because there are a lot of different sources of edges, and we have no guarantee that the edges being used in the scoring process are the right ones.

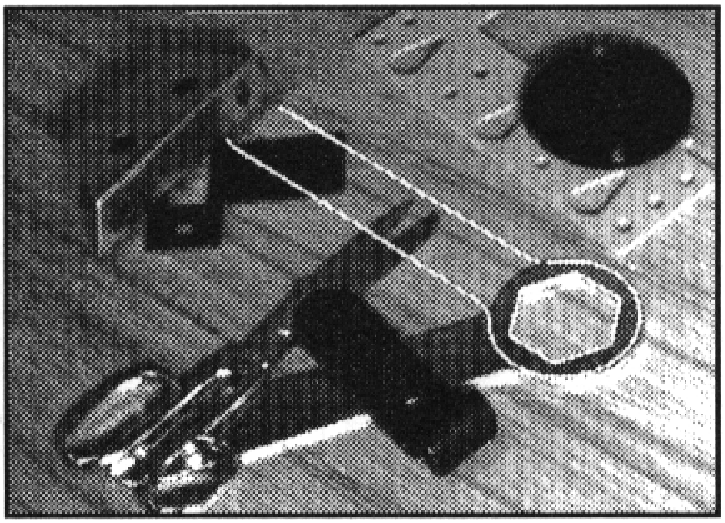
A poor pose estimate can lead to silhouette edges on the backprojected object lying a long way from the actual image edges. For example, if we have an object that projects to a long thin region, like a spanner, and we estimate its image plane orientation using one end, the backprojected edges at the other end may lie a long way from the image edge points we are interested in (figure 19.7). This is an example of error propagation — essentially, the camera estimate is good for features nearby those used to obtain it, but increasingly bad for those a long way away. Several fixes are possible:

- **Maximize over the pose estimate:** The situation can sometimes be improved by maximising the verification score with respect to the pose. This doesn't always work; if the original pose estimate is poor, the object may indeed lie close to edges, but to the wrong edges — think about an attempt to verify the presence of an object on a textured background. Furthermore, the optimisation can be hard, particularly if the `nearby` test is a threshold on distance.

- **Count only edges for which the camera estimate is reliable:** To our knowledge, this has never been tried in practice. The advantage of the approach is that it should deal with issues like the example of figure 19.7 relatively easily; the disadvantage is that one may not use a large portion of the object in the verification process, meaning that there is a very good prospect of false positives.

Counting the wrong edges in the score is another major source of difficulties. In textured regions, there are many edge points in small collections. The whole point of verification is to use the model to link up image evidence that is too hard to gather together in the hypothesis formation stage, so we can't exclude small groups of edge points from the score. This means that, in highly textured regions, it is possible to get high verification scores for almost any model at almost any pose (for example, see figure 19.7); notice that the fact that we are counting similarity in edge orientation in the verification score hasn't made any difference here.

We can tune the edge detector to smooth texture heavily, in the hope that textured regions will disappear. This is a dodge, and a dangerous one, because it usually affects the contrast sensitivity so that the objects disappear, too. It can be made to work acceptably, and is widely used.



0.99

Figure 19.7. Edge orientation can be a deceptive cue for verification, as this figure (reproduced from [**?**], p. 108 - as OUP will ideally give permission) illustrates. The edge points marked on the image come from a model of a spanner, recognised and verified with 52 % of its outline points matching image edge points with corresponding orientations. Unfortunately, the image edge points come from the oriented texture on the table, not from an instance of the spanner. As the text suggests, this difficulty could be avoided with a much better description of the spanner's interior as "untextured", which would be a poor match to the oriented texture of the table.

19.5.2 Similarity in Texture, Pattern and Intensity

If we score edge matches, then texture is not much more than a nuisance. However, some objects have quite distinctive textures — for example, camouflage paint — and this should probably be used. We could describe model regions using texture descriptors (like the statistics of filter outputs in a region in chapter **??**), and then compare those descriptors with the image. The comparison would need to estimate the significance of the difference between the image and the backprojected object regions. The most promising approach appears to be comparing the probability of obtaining the image region covered by the object region by drawing a texture from the object family, with the probability of obtaining this texture when an object is absent.

Comparing silhouette edges ignores a great deal of useful information. If objects are patterned — meaning there are large scale coloured regions like the markings on a soda can — we could compare backprojected pattern edges as well. A more sophisticated approach would compare backprojected pattern regions with the image regions by using texture descriptors (which should agree on the absence of texture) and perhaps descriptions of hue and saturation (some examples of this approach appear in []).

We do not usually have enough information about illumination to predict object intensities. As a result, intensities tend to be ignored in practice in verification. This is a mistake. Many differences in intensity patterns seldom, if ever, come from light sources — for example, only very strange light sources like movie projectors generated textured intensity patterns. This suggests that one could probably obtain a verification score by comparing differences in absolute intensity with differences that have arisen in practice, and differences that could not arise in practice.

19.5.3 Example: Bayes Factors and Verification

Verification can be thought of as model selection. We are comparing the model that the pattern in a region is a result of the presence of some object with the model that it appeared without the object there. We could set this up as a Bayesian model selection exercise; this gives a useful way to think about using texture and intensity information in verification.

Assume we must determine whether a camouflaged aircraft is present in a calibrated perspective camera at some hypothesized pose. We would like to do this using texture and illumination features. Camouflage painters use a system of marks that is quite stylised but is not the same from aircraft to aircraft.

A natural set of features to use to describe the texture is to use filters whose scale depends on object pose — which we shall write as $\boldsymbol{\theta}$, and take histograms of their squared output; we might also apply larger scale filters to the filter outputs, and histogram their outputs, too. The domain in which to compute these features follows from the object pose — we could predict it by backprojection. Assume we have a set of features, $\boldsymbol{y}(\boldsymbol{\theta})$; we can now take many pictures of the relevant aircraft, to estimate a conditional distribution $f(\boldsymbol{y}|\boldsymbol{\theta}, aircraft)$ (using any attractive density estimation technique —- see chapter **??**).

Similarly, we can measure a variety of other cases, too, say: $f(\boldsymbol{y}|\boldsymbol{\theta}, bush)$, $f(\boldsymbol{y}|\boldsymbol{\theta}, tarmac)$, $f(\boldsymbol{y}|\boldsymbol{\theta}, grass)$ and $f(\boldsymbol{y}|\boldsymbol{\theta}, building)$, which might be an exhaustive set of cases for views around an airfield. In this case, $\boldsymbol{\theta}$ determines the domain on which the features are measured and the scale of the filters.

The Bayes factor compares posterior likelihoods for models. Thus, we would like to test:

$$\frac{f(aircraft|\boldsymbol{y})}{f(\text{anything else}|\boldsymbol{y})}$$

If this fraction is very large, that suggests that we have strong evidence that an aircraft is present; if it is very small, we have strong evidence that it is absent. We can rewrite the fraction as:

$$\frac{\int f(\boldsymbol{y}|\boldsymbol{\theta}, aircraft)\pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{aircraft}}{\int f(\boldsymbol{y}|\boldsymbol{\theta}, \text{anything else})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{\text{anything else}}}$$

and the denominator is:

$$\int \left(\begin{array}{l} f(\boldsymbol{y}|\boldsymbol{\theta}, bush)\pi(bush|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, grass)\pi(grass|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, runway)\pi(runway|\text{anything else})+ \\ f(\boldsymbol{y}|\boldsymbol{\theta}, buildings)\pi(buildings|\text{anything else}) \end{array} \right) \pi(\boldsymbol{\theta})d\boldsymbol{\theta}\pi_{\text{anything else}}$$

where $\pi(bush|\text{anything else})$ measures how probable it is that a region that is not an aircraft is bush — we might estimate this from area measurements in aerial photographs — $\pi_{\text{anything else}}$ is an estimate of how reliable our hypothesis generating process is and $\pi(\boldsymbol{\theta})$ is an estimate of the probability of the pose, given our estimate — this might be a fairly narrow distribution, peaked about the estimate from our hypothesis generating process.

Notice the advantages of this process: we are accounting for the uncertainty in our pose estimate, essentially by adding in weighted versions of the match quality at nearby poses; we are verifying on the extent to which the pattern looks like an aircraft and unlike other things in a fairly flexible way — if the camouflage painters change their work practices, then we know what to do; and we are accounting for the different types of clutter individually.

19.6 Application: Registration in Medical Imaging Systems

There are numerous problems where pose is far more important than recognition. Many recognition algorithms were designed in the expectation that a selection of industrial parts would be scattered in a bin or on a table; it turns out that production engineers are quite careful to ensure that their parts do not get mixed up, but would often like very accurate measurements of pose. Medical applications are similar, in that it is usually known *what* is being looked at, but there is a crucial need for an accurate measurement of *where* it is.

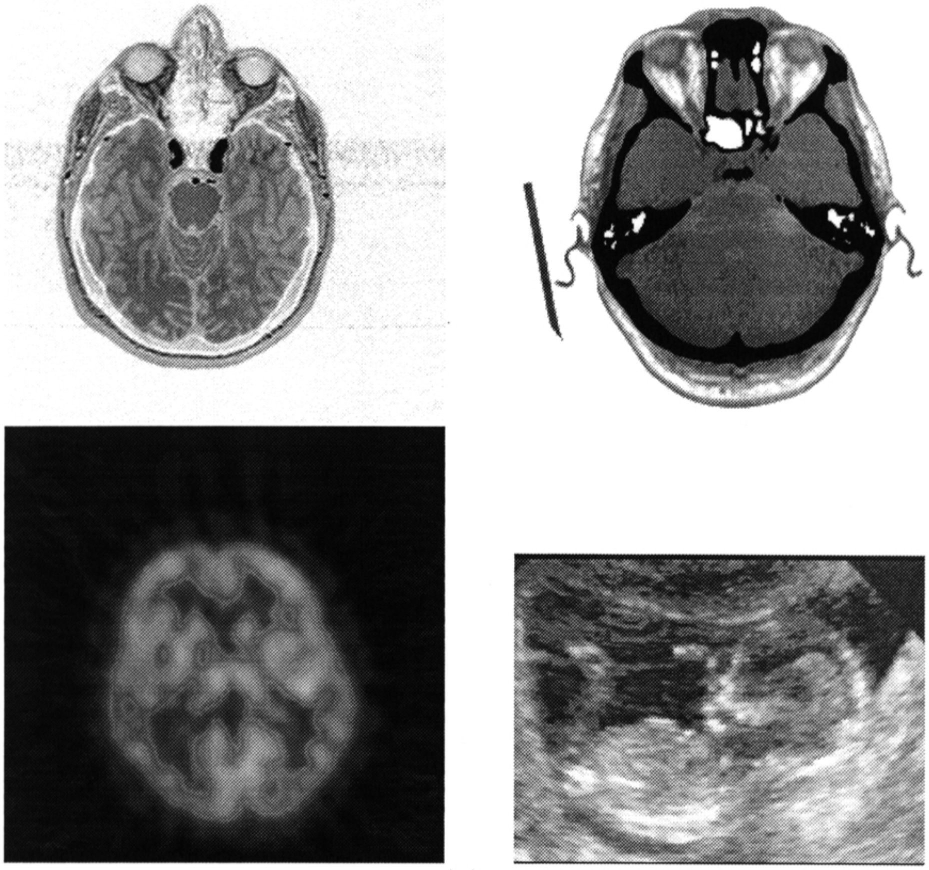


Figure 19.8. Images obtained with four different imaging modes. On the top left, an MRI image of a cross-section of the skull; on the top right, a CTI image of a cross-section of the skull; on the bottom left, an NMI image of a brain; and on the bottom right, a USI image of a foetus in a womb. Notice how each modality shows different detail in different ways; there is high-resolution detail of the brain in the MRI image and of the skull in the CTI image. The NMI image is at low resolution, but (in fact) reflects function, because regions that respond strongly have taken up some reagent. Finally, the USI image has a significant noise component, but shows details of soft tissue — you should be able to see a leg, the body, the head and a hand of the foetus. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 6, in fervent hope that permission will be granted.*

19.6.1 Imaging Modes

There are a variety of imaging technologies available, including **magnetic resonance imaging** (MRI) which uses magnetic fields to measure the density of protons, and is typically used for descriptions of organs and soft tissue; **computed tomography imaging** (CTI or CT), which measures the density of X-ray absorbtion, and is typically used for descriptions of bones; **nuclear medical imaging** (NMI), which measures the density of various injected radio-active molecules, and is typically used for functional imaging; and **ultra-sound imaging**, which measures variations in the speed of ultrasound propagation, and is often used to obtain information about moving organs (figure 19.8 illustrates these modes). All of these techniques can be used to obtain slices of data, which allow a 3D volume to be reconstructed. A standard problem is to segment these volumes into various structures; figure 19.9 shows an MRI image with the brain, brain ventricles and tumour segmented. Since tumours are essentially fixed with respect to the skull and skin, this data gives us the position of the tumour with respect to the head.

Registration in medical imaging is almost always a 3D from 3D problem, and the only transformations to care about are 3D rotations and translations. Geometric hashing is the dominant mechanism, because it can be used to search correspondences efficiently. The literature differs largely in the matter of what data is used.

19.6.2 Applications of Registration

In **brain surgery** applications, surgeons are attempting to remove tumours while doing the minimum of damage to a patient's faculties. We shall show examples due to Grimson *et al.*, []. The general approach is to obtain images of the patients brain, segment these images to show the tumour, and then display the images to the surgeon. The display is overlaid on pictures of the patient on the table, obtained using a camera near the surgeon's view, to cue the surgeon to the exact position of the tumour. Various methods exist for attaching functional tags to the image of the brain — usually, one stimulates a region of the brain, and watches to see what happens — and this information can also be displayed to the surgeon, so that the impact of any damage done can be minimized. The problem here is pure pose estimation; we need to know the pose of the brain image and the brain measurements with respect to the person on the table.

Reconstructive surgery offers similar applications. For example, in facial reconstruction, in a planning phase, surgeons can be allowed to work out a sequence of activities on a visualisation of a patient's skull. The results of this visualisation will need to be displayed to the surgeons when they operate, again registered to a view of the patient.

Diagnostic applications include creating 3D visualisations to display results from many different imaging modes. For example, a surgeon may have images from MRI — which quite often has relatively high resolution — and PET — which are linked to functional properties. It is natural to wish to fuse these images, and so they need to be registered.

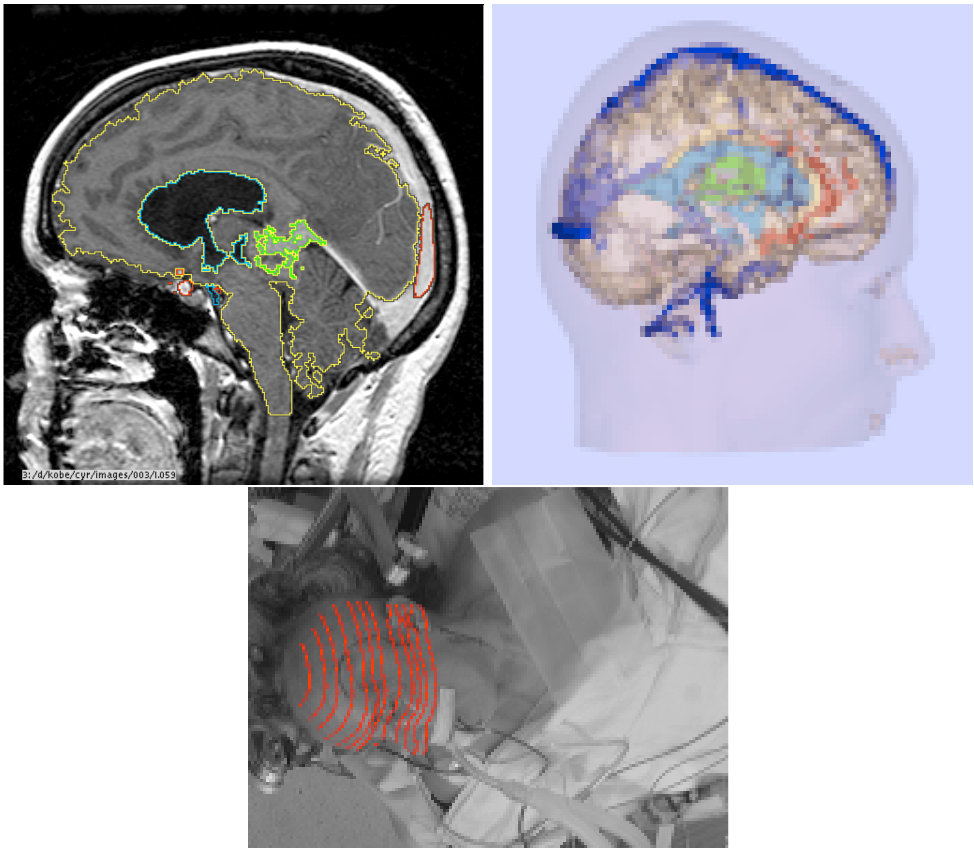


Figure 19.9. On the **top left**, a single slice of MRI data, with an automatically acquired segmentation overlaid. The segmentation outlines the brain, vacuoles within the brain, and the tumour. MRI produces a sequence of slices, which yield a volume model; a view of a segmented volume model, with different colours showing different regions, is shown at the **top right**. Once this data is obtained, it is registered to a patient lying on a table. Registration is obtained using depth data, measured by a laser ranger; the **bottom** figure shows a camera view of a patient with laser ranger data overlaid. *Data obtained from Eric Grimson's web site* `http://www.ai.mit.edu/people/welg/welg.html`*, in the fervent hope that permission will be granted*

19.6.3 Geometric Hashing Techniques in Medical Imaging

The main differences between algorithms in applications is the type of measurement used for geometric hashing. We discuss a few cases below.

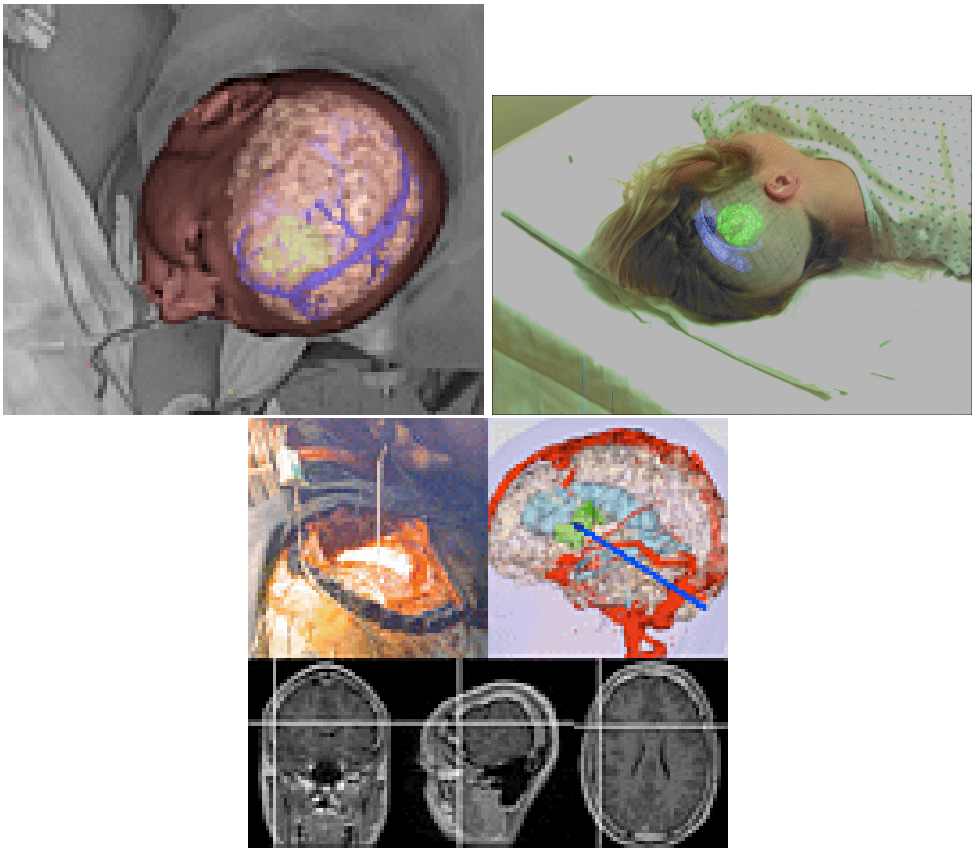


Figure 19.10. The figure on the **top left** shows skin data overlaid on a view of a patient to indicate the success of the registration process. Once data is registered to a patient, a number of uses are available. At the **top right**, we see a view of a patient on an operating table with MRI imagery showing part of the brain and a tumour overlaid for the surgeon's information. At the **bottom**, we see imagery obtained by registering the position of a surgical instrument with the MRI data set — this means that the position of the surgical instrument can be displayed to the surgeon, however deep the instrument is within tissue. *Data obtained from Eric Grimson's web site* `http://www.ai.mit.edu/people/welg/welg.html`*, in the fervent hope that permission will be granted*

Point Correspondences

We have seen how to search point correspondences. For example, head MRI data can be registered to a patient's head on a table by obtaining 3D measurements of the head with a laser ranger on the operating table, and using these to register with the skin points on the MRI data. We can hash pairs of skin points from the MRI on their distance apart and the angle between their normals, and then query with pairs of points from the laser data. Pairs with similar distances apart and similar angles could correspond. With a corresponding pair, we can estimate pose, and then check the total error of this pose estimate. While there is no true correspondence — neither the skin points in the MRI image nor the laser ranger data consists of isolated points; they are samples from surfaces — the sampling is sufficiently dense that it is possible to obtain good initial hypotheses about pose. If the error in the registration is then measured sensibly — counting only error components normal to the surface to avoid being put off by small localisation errors — an excellent pose estimate can be obtained by minimising the error. Grimson is the main proponent of such systems, one of which is illustrated in figure 19.10; details appear in [].

Curves

Curves can be used to drive geometric hashing, too. In this case, we fit a surface to the dataset, and then mark significant curves on this surface. Using parabolic curves is impractical, because some datasets have many flat regions; Ayache [] has successfully used curves where the maximum normal curvature is an extremal along the curves of maximun normal curvature on the surface (figures 19.11 and 19.12). Whatever curve is used, at any point on a curve we have a complete 3D frame (the Serret-Frenet frame of appendix **??**), and we can use this frame as a canonical frame to obtain measurements for geometric hashing. Natural choices include the curvature and torsion of the curve, and the angle between the curve normal and the surface normal.

Frame Pairs

Given two coordinate frames, the transformation from one to the other is invariant to a shared transformation, and so can be used to supply indexes for hashing. We call this cue a **frame pair**. A natural way to obtain these pairs is to fit a surface to the dataset, identify significant curves or points, and then use local frames. For a significant point on a surface — for example, an umbilic point — a frame can be obtained from the normal and the two directions of extremal curvature. For any point on a significant curve, we can use either the Serret Frenet frame, or the frame on the surface (or compare the two frames). Ayache and his group have emphasized the use of curves and of frame pairs; details appear in [].

19.7 Curved Surfaces and Alignment

Curved surfaces can be aligned, too. The hypothesis generating process can be more complicated, but rendering and verification are straightforward generalisations.

The natural strategy is to find frame bearing groups that behave like points; for example, if a curved surface has points painted on it, or if three surfaces meet discontinuously at points, the hypothesis generating process behaves like those described above. A geometric model of the surface can then be projected into the image, and used to verify as below.

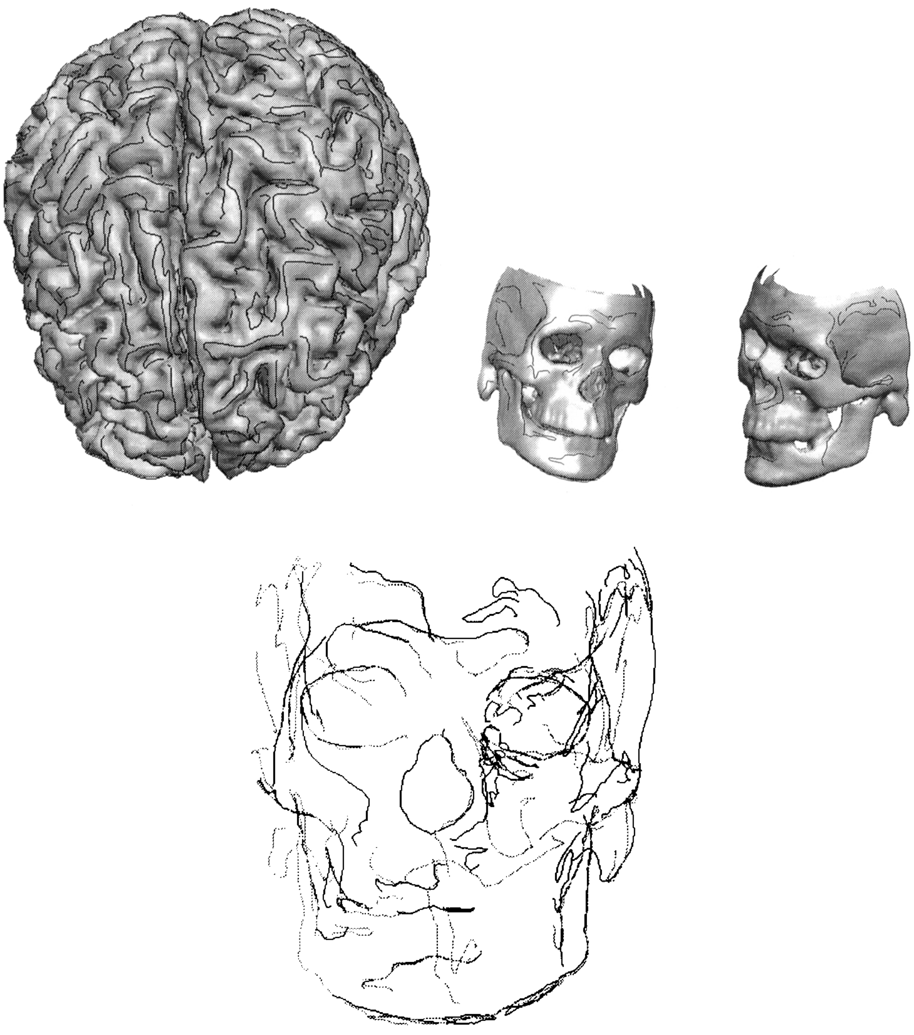


Figure 19.11. At the **top left**, ridge curves — curves where the maximum normal curvature is extremal in its direction — obtained from a spline surface fit to an image of a brain. Notice that there is a profusion of such curves, which can be used as a rich source of alignment information. At the **top right**, ridge curves for two distinct images of a skull; and at the **bottom**, ridge curves aligned using an algorithm structured around geometric hashing. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 20,30, in fervent hope that permission will be granted.*

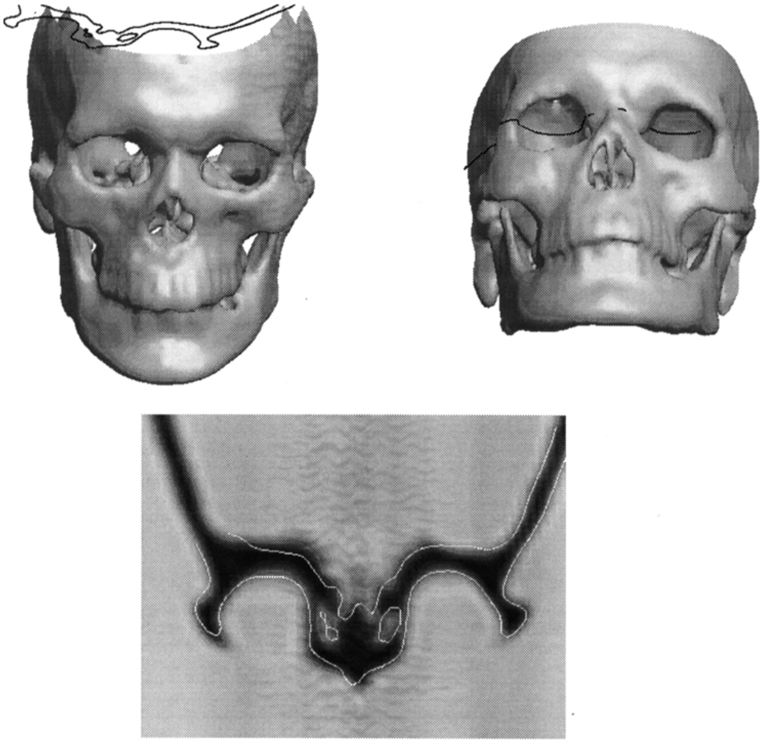


Figure 19.12. Curve matching by geometric hashing can be used to match slices to datasets, too. On the top left, we see a surface corresponding to a skull, extracted from a CTI image. The curve is obtained from another slice of data. On the top right, we see this curve aligned with the surface, using geometric hashing (in this case, we would fill the hash table with curves corresponding to each plane section of the surface and hash on the plane). Below, we see the curve superimposed on the plane extracted from the 3D image. *Data obtained from Nicholas Ayache's paper, "Medical Computer Vision, Virtual Reality and Robotics - Promising Research Tracks", page 33, in fervent hope that permission will be granted.*

A more difficult approach sees alignment as minimisation, rather like the linear combinations of models approach discussed above. In this approach, the outline of the surface is predicted as a function of the pose of the surface. We can adopt as an objective function the sum of the minimum distances of selected edge points from this outline, and minimize the objective function over pose, as figure 19.13 illustrates. The mechanics of predicting outline curves is simplified for algebraic surfaces; all examples in the literature use algebraic surfaces as an example for this reason. The details appear in []. Algebraic surfaces are so rigid that a single outline completely determines the surface geometry; masochists can look up the details in [].

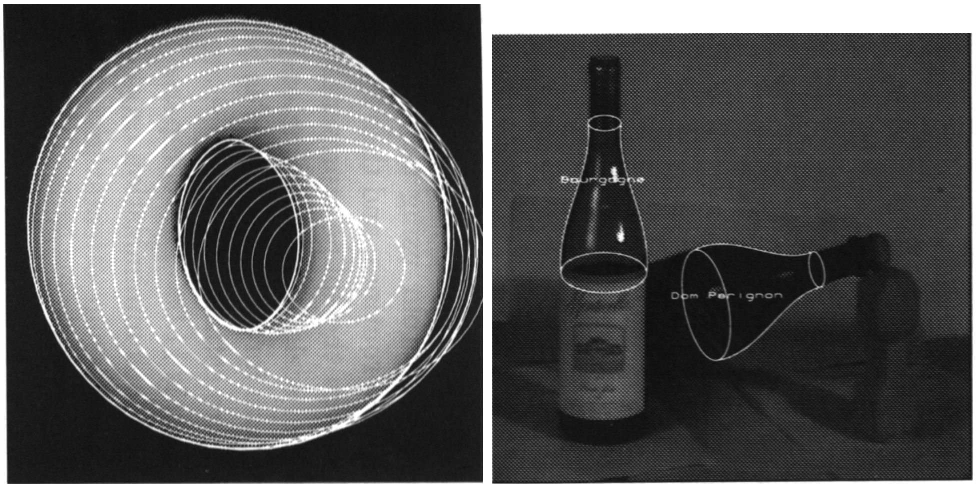


Figure 19.13. An algebraic surface is viewed in a calibrated perspective camera. The contour generator is clearly an algebraic curve (because we can write down a set of polynomial equations that it satisfies) and the outline is also an algebraic curve. This curve is a function of the pose of the surface. The figure on the left shows a family of curves obtained by overlaying the outline of a surface on an image, and obtaining pose by minimizing the sum of the minimum distances between selected edge points and the outline. The curves come from different points in the minimization processs. The figure on the right shows two different algebraic surfaces aligned successfully with image contours; the surfaces used identify two different bottles.

19.8 Discussion

Typically, systems built around the algorithms described in this chapter can recognise small numbers of objects in quite cluttered scenes. They are important because pose recovery and registration is useful in applications, and because their weaknesses point out important issues in recognition.

Alignment is **quite general** because for most conceivable images of most conceivable objects, some form of camera consistency constraint applies and can be exploited. It is also quite **noise resistant** — meaning that we can find objects even in heavily cluttered images — because relatively little image evidence needs to be collected to construct an object hypothesis. Testing a large hypothesis robustly tends to be easier than assembling a large hypothesis, because the hypothesis under test gives very strong constraints on what image evidence can contribute to a decision (the noise behaviour of some alignment algorithms has been studied in detail []). However, alignment **scales poorly** with increasing numbers of models.

Linear growth in the number of models occurs because the modelbase is *flat* - there is no hierarchy, and every model is treated in the same way.

Recognition systems based on algebraic invariants have quite limited application, because of the requirements for special features like lines and conics and because relatively few cases of 3D objects are manageable. There are some features of these systems that are worth further thought, however.

Choice of measurements: We saw that there were situations where many invariant measurements are available, and it was not clear how to choose which ones should be used. Some properties may be highly distinctive. Other properties may be shared by many objects, or may arise easily from clutter. Clearly, we should like to measure distinctive properties; determining which properties are distinctive and how they should be used requires a more extensive theory of measurement than we have seen at present.

Modelling with pictures: One substantial advantage of using invariants to model objects is that pictures can serve as an object model. This is a feature we would want to preserve.

Segmentation: As the figures show, systems built using these ideas function quite well in worlds containing substantial amounts of clutter. This probably has to do with the way in which they confine attention to particular image structures. In the case of the system described above, all image information that doesn't form edges is discarded; then all edges that do not form a reasonably long line are discarded; then all lines that don't join up into a sequence of five lines are discarded; then all of these assemblies without the right projective invariants are ignored; and only assemblies that satisfy this sequence of tests are accepted as hypotheses. This is a segmentation mechanism that is directed by the contents of the model base. It consists of stages, which run (rather roughly) from the general (edges) to the particular (assemblies of five lines that have the right projective invariants). While it is crude, it is powerful because it is hard to generate groups of the type sought by accident.

None of the algorithms described handle objects with **internal degrees of freedom** well. There are essentially two alternatives: decomposing the object into rigid parts, recognising the parts and then assembling them — which gets us into completely new territory (see chapter **??**), because the parts may be hard to recognise individually; or incorporating the parameter into the matching process, either by minimizing the verification score over the parameter or by folding the parameter into the camera calibration process or the indexing process — which will give difficulties for more than a small number of parameters.

None of the algorithms described recognize objects at any level of **abstraction**. A typical attack on this problem is to define parametric families of models and extend the consistency argument to include the model parameter — this is usually done in the context of linear combinations of models, but each method allows it. This line of attack completely misses the point of abstraction.

Each algorithm attempts to get enough information to perform verification with as little trouble as possible, while trying to reduce the number of spurious verification attempts. This dependency on *image level* verification is inconsistent with model abstraction — we could not verify that a picture contained a fish by looking at pixel values or edges if we did not have exact details of its species, configuration, and the like. Image level verification is also an "admission of guilt" — we are backprojecting the model to decide which components of the image come from the object because we don't know a better way of assembling the evidence.

One could avoid this problem by denying the significance of abstraction. This is a bit like pretending the sun doesn't come up in the mornings. Most applications of recognition desperately need abstraction. For example, if we want to search the Internet for pictures of the Pope, we don't want to have to know his exact geometry. Similarly, if we want to deploy our automatic motor car on real roads, it has to be able to decide what it should swerve to avoid and what it may run down.

There is an attempt to repair this difficulty in the alignment literature, by replacing features (like points and lines) with *tokens* (where a token might be a hairy patch, or an eye, or something of the sort) to models containing tokens. Here, the abstraction is in the token [**?**; **?**]. This dodge would work for any of the other algorithms we have described as well. It is a dodge, precisely because it focusses on the detailed geometric relations between the tokens: what if the bits move around, as in a face? or a person? The answer is probably that relationships should be tokens, too, and that collections of tokens should be tokens. We shall explore the implications and difficulties of this old idea in chapters **??**.

The main role of verification is to find evidence for an hypothesis that could not be collected in other ways; since collecting evidence is poorly understood, current recognition systems work well when verification works well, and badly otherwise. Verification will work well when sufficient evidence is used, and scored appropriately. Unfortunately, it is difficult to translate these platitudes into algorithms. For a topic that is so central to the performance of recognition systems, verification has been extremely poorly studied. There is no organised literature, although there are some results on the effects of hypothesis errors ([**?**]).

Verification based on generic evidence — say, edge points — has the difficulty that we cannot tell which evidence should be counted. Similarly, if we use specific evidence — say a particular camoflage pattern — we will have problems with abstraction. Template matching and appearance based vision, which we discuss in greater detail in chapter **??**, can be seen as mechanisms to involve more kinds of evidence in the verification process. We discuss how to weigh this evidence in chapter **??**.

Assignments

Exercises

- Assume that we are viewing objects in a calibrated perspective camera, and wish to use a pose consistency algorithm for recognition.

    1. Show that three points is a frame group.

2. Show that a line and a point is *not* a frame group.
3. Explain why it is a good idea to have frame groups composed of different types of feature.
4. Is a circle and a point not on its axis a frame group?

- We have a set of plane points $\boldsymbol{P}_j$; these are subject to a plane affine transformation. Show that

$$\frac{\det\left[\boldsymbol{P}_i \boldsymbol{P}_j \boldsymbol{P}_k\right]}{\det\left[\boldsymbol{P}_i \boldsymbol{P}_j \boldsymbol{P}_l\right]}$$

  is an affine invariant (as long as no two of i, j, k and l are the same, and no three of these points are collinear).

- Use the result of the previous exercise to construct an affine invariant for:

  1. Four lines;
  2. Three coplanar points;
  3. A line and two points (these last two will take some thought).

- In chamfer matching, at any step, a pixel can be updated if the distances from some or all of its neighbours to an edge are known; Borgefors counts the distance from a pixel to a vertical or horizontal neighbour as 3 and to a diagonal neighbour as 4 to ensure the pixel values are integers. Why does this mean $\sqrt{2}$ is approximated as $4/3$? Would a better approximation be a good idea?

- One way to improve pose estimates is to take a verification score, and then optimize it as a function of pose. We said that this optimization could be hard, particularly if the test to tell whether a backprojected curve was close to an edge point was a threshold on distance. Why would this lead to a hard optimisation problem?

- We said that for an uncalibrated affine camera viewing a set of plane points, the effect of the camera can be written as an unknown plane affine transformation. Prove this. What if the camera is an uncalibrated perspective camera viewing a set of plane points?

- Prepare a summary of methods for registration in medical imaging other than the geometric hashing idea we discussed. You should keep practical constraints in mind, and you should indicate which methods you favour, and why.

- Prepare a summary of non-medical applications of registration and pose consistency.

Programming Assignments

1. Representing an object as a linear combination of models is often represented as abstraction, because we can regard adjusting the coefficients as obtaining the same view of different models. Furthermore, we could get a parametric family of models by adding a basis element to the space. Explore these ideas by building a system for matching rectangular buildings where the width, height and depth of the building are unknown parameters. You should extend the linear combinations idea to handle orthographic cameras — this involves constraining the coefficients to represent rotations.

Chapter 20

FINDING TEMPLATES USING CLASSIFIERS

There are a number of important object recognition problems that involve looking for image windows which have a simple shape and stylised content. For example, frontal faces appear as oval windows, and (at a coarse scale) all faces look pretty much the same — a dark horizontal bar at the eyes and mouth, a light vertical bar along the nose, and not much texture on the cheeks and forehead. As another example, a camera mounted on the front of a car will always see relevant stop signs as having about the same shape and appearance.

This suggests a view of object recognition where we take all image windows of a particular shape and test them to tell if the relevant object is present. If we don't know how big the object will be, we can search over scale, too; if we don't know its orientation, we might search over orientation as well, etc. Generally, this approach is referred to as **template matching**. There are some objects that can be found very effectively with a template matcher. Faces and roadsigns are important examples. Secondly, while many objects appear to be hard to find with simple template matchers (it would be hard to find a person this way, because the collection of possible image windows that represent a person is immense), there is some evidence that reasoning about *relations* between many different kinds of templates can be an effective way to find objects. In chapter **??**, we explore this line of reasoning further.

The main issue to study in template matching is how one builds a test that can tell whether an oval represents a face or not. Ideally, this test will be obtained using a large set of examples. The test is known as a **classifier** — a classifier is anything that takes a feature set as an input and produces a class label. In this chapter, we describe a variety of techniques for building classifiers, with examples of their use in vision applications. We first present the key ideas and terminology used (section 20.1); we then show two successful classifiers built using histograms (section 20.2); for more complex classifiers, we need to choose the features a classifier should use, and we discuss two methods in (section 20.3). Finally, we describe two different methods for building classifiers with current applications in vision. Section **??** is an introduction to the use of neural nets in classification; and section 20.5 describes a very useful classifier known as a support vector machine.

20.1 Classifiers

Classifiers are built by taking a set of labelled examples and using them to come up with a rule that will assign a label to any new example. In the general problem, we have a training data set $(\mathbf{x}_i, y_i)$; each of the $\boldsymbol{x}_i$ consists of measurements of the properties of different types of object, and the y_i are labels giving the type of the object that generated the example. We know the relative costs of mislabelling each class, and must come up with a rule that can take any plausible $\boldsymbol{x}$ and assign a class to it.

The cost of an error significantly affects the decision that is made. In section 20.1.1, we study this question. It will emerge that the probability of a class label given a measurement is the key matter. In section 20.1.2, we discuss methods for building appropriate models in a general way. Finally, we discuss how to estimate the performance of a given classifier (section 20.1.5).

20.1.1 Using Loss to Determine Decisions

The choice of classification rule must depend on the cost of making a mistake. For example, doctors engage in classification all the time — given a patient, they produce the name of a disease. A doctor who decided that a patient suffering from a dangerous and easily treated disease is well, is going to have problems. It would be better to err on the side of misclassifying healthy patients as sick even if doing so involves treating some healthy patients unnecessarily.

The cost depends on what is misclassified to what. Generally, we write outcomes as $(i \to j)$, meaning that an item of type i is classified as an item of type j. Each outcome has its own cost, which is known as a **loss**. Hence, we have a loss function which we shall write as $L(i \to j)$, meaning the loss incurred when an object of type i is classified as having type j. Since losses associated with correct classification should not affect the design of the classifier, $L(i \to i)$ must be zero; but the other losses could be any positive numbers.

The **risk function** of a particular classification strategy is the expected loss when using it, as a function of the kind of item. The **total risk** is the total expected loss when using the classifier. Thus if there were two classes, the total risk of using strategy s would be:

$$R(s) = Pr\left\{1 \to 2|\text{using } s\right\} L(1 \to 2) + Pr\left\{2 \to 1|\text{using } s\right\} L(2 \to 1)$$

The desirable strategy is one that minimizes this total risk.

Building a Two Class Classifier that Minimizes Total Risk

Assume that the classifier can choose between two classes, and we have a known loss function. There is some boundary in the feature space — which we call the

decision boundary — such that points on one side belong to class one and points on the other side to class two.

We can resort to a trick to determine where the decision boundary is. It must be the case that, *for points on the decision boundary of the optimal classifier*, either choice of class has the same expected loss — if this wasn't so, we could obtain a better classifier by always choosing one class (and so moving the boundary). This means that, for measurements on the decision boundary, choosing class one yields the same expected loss as choosing class two.

A choice of class one for a point $\boldsymbol{x}$ at the decision boundary yields an expected loss

$$\begin{aligned} P\left\{\text{class is } 2|\boldsymbol{x}\right\} L(2 \to 1) + P\left\{\text{class is } 1|\boldsymbol{x}\right\} L(1 \to 1) &= P\left\{\text{class is } 2|\boldsymbol{x}\right\} L(2 \to 1) + 0 \\ &= p(2|\boldsymbol{x}) L(2 \to 1) \end{aligned}$$

(you should watch the one's and two's closely here). Similarly, a choice of class two for this point yields an expected loss

$$P\left\{\text{class is } 1|\boldsymbol{x}\right\} L(1 \to 2) = p(1|\boldsymbol{x}) L(1 \to 2)$$

and these two terms must be equal. This means our decision boundary consists of the points $\boldsymbol{x}$ where

$$p(1|\boldsymbol{x}) L(1 \to 2) = p(2|\boldsymbol{x}) L(2 \to 1)$$

We can come up with an expression that is often slightly more practical, by using Bayes' rule. Rewrite our expression as

$$\frac{p(\boldsymbol{x}|1)p(1)}{p(\boldsymbol{x})} L(1 \to 2) = \frac{p(\boldsymbol{x}|2)p(2)}{p(\boldsymbol{x})} L(2 \to 1)$$

and clear denominators to get

$$p(\boldsymbol{x}|1)p(1)L(1 \to 2) = p(\boldsymbol{x}|2)p(2)L(2 \to 1)$$

This expression identifies points $\boldsymbol{x}$ on a class boundary; we now need to know how to classify points off a boundary.

At points off the boundary, we must choose the class with the *lowest* expected loss. Recall that, if we choose class two for a point $\boldsymbol{x}$, the expected loss is

$$p(1|\boldsymbol{x}) L(1 \to 2)$$

etc. This means that we should choose class one if

$$p(1|\boldsymbol{x}) L(1 \to 2) > p(2|\boldsymbol{x}) L(2 \to 1)$$

and class two if

$$p(1|\boldsymbol{x}) L(1 \to 2) < p(2|\boldsymbol{x}) L(2 \to 1)$$

A Classifier for Multiple Classes

From now on, we shall assume that $L(i \to j)$ is zero for $i = j$ and one otherwise – that is, that each outcome has the same loss. In some problems, there is another option, which is to refuse to decide which class an object belongs to. This option involves some loss, too, which we shall assume to be $d < 1$ (if the loss involved in refusing to decide is greater than the loss involved in any decision, then we'd never refuse to decide).

For our loss function, the best strategy — which is known as the **Bayes classifier** — is given in algorithm 1. The total risk associated with this rule is known as the **Bayes risk** — this is the smallest possible risk that we can have in using a classifier. It is usually rather difficult to know what the Bayes classifier — and hence the Bayes risk — is, because the probabilities involved are not known exactly. In a few cases it is possible to write the rule out explicitly. One way to tell the effectiveness of a technique for building classifiers is to study the behaviour of the risk as the number of examples increases — for example, one might want the risk to converge to the Bayes risk in probability if the number of examples is very large. The Bayes risk is seldom zero, as figure 20.1 illustrates.

For a loss function

$$L(i \to j) = \begin{cases} 1 & i \neq j \\ 0 & i = j \\ d < 1 & \text{no decision} \end{cases}$$

the best strategy is

- *if $Pr\{k|\mathbf{x}\} > Pr\{i|\mathbf{x}\}$ for all i not equal to k, and if this probability is greater than $1 - d$, choose type k*
- *if there are several classes $k_1 ... k_j$ for which $Pr\{k_1|\mathbf{x}\} = Pr\{k_2|\mathbf{x}\} = ... = Pr\{k_j|\mathbf{x}\} > Pr\{i|\mathbf{x}\}$ for all i not in $k_1, ..k_j$, choose uniformly and at random between $k_1, ..k_j$*
- *if for all k we have $Pr\{k|\mathbf{x}\} > Pr\{i|\mathbf{x}\} \leq 1 - d$, refuse to decide.*

Algorithm 20.1: *The Bayes classifier classifies points using the posterior probability that an object belongs to a class, the loss function, and the prospect of refusing to decide.*

20.1.2 Overview: Methods for Building Classifiers

Usually, we do not know $Pr\{\mathbf{x}|k\}$ exactly — which are often called **class-conditional densities** — or $Pr\{k\}$, and must determine a classifier from an example data set. There are two rather general strategies:

- **Explicit probability models:** we can use the example data set to build a probability model (of either the likelihood or the posterior, depending on

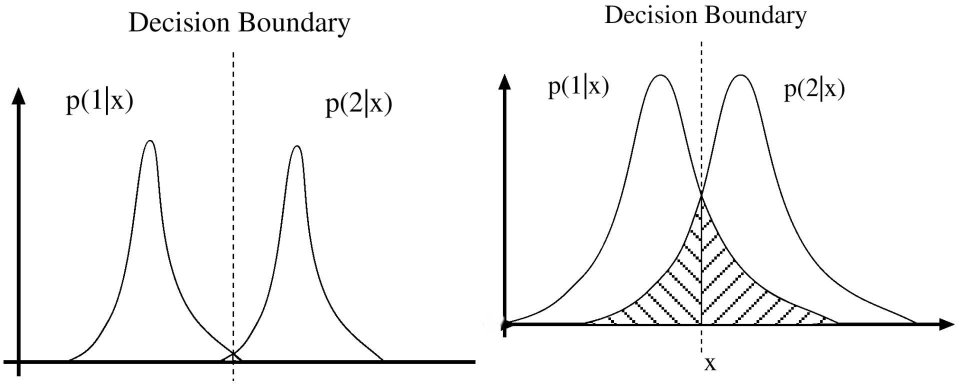


Figure 20.1. This figure shows typical elements of a two class classification problem. We have plotted $p(\text{class}|x)$ as a function of the feature x. Assuming that $L(1 \to 2) = L(2 \to 1)$, we have marked the classifier boundaries. In this case, the Bayes risk is the sum of the amount of the posterior for class one in the class two region and the amount of the posterior for class two in the class one region (the hatched area in the figures). For the case on the left, the classes are well separated, which means that the Bayes risk will be small; for the case on the right, the Bayes risk is rather large.

taste). There are a very wide variety of ways of doing this, some of which we shall see in the following sections. In the very simplest case, we know that the class-conditional densities come from some known parametric form of distribution. In this case, we can compute estimates of the parameters from the data set, and plug these estimates into the Bayes rule. This strategy is often known as a **"plug-in" classifier** (section **??**). This approach covers other parametric density models and other methods of estimating parameters. One subtlety is that the "best" estimate of a parameter may not give the best classifier, because the parametric model may not be correct. Another subtlety is that a good classifier may be obtained using a parametric density model that is not a very accurate description of the data (see figure 20.2). In many cases, it is hard to obtain a satisfactory model with a small number of parameters. More sophisticated modelling tools (such as neural nets, which we deal with in some detail in section 20.4) provide very flexible density models that can be fitted using data.

- **Determining decision boundaries directly:** Quite bad probability models can produce good classifiers, as figure 20.2 indicates. This is because the decision boundaries are what determine the performance of a classifier, not the details of the probability model (the main role of the probability model in the Bayes classifier is to identify the decision boundaries). This suggests that we might ignore the probability model, and attempt to construct good decision boundaries directly. This approach is often extremely successful; it is particularly attractive when there is no reasonable prospect of modelling the data source. One strategy assumes that the decision boundary comes from one or another class, and constructs an extremisation problem to choose the best element of that class. A particularly important case comes when the data is **linearly separable** — which means that there exists a hyperplane with all the positive points on one side and all the negative points on the other — and thus that a hyperplane is all that is needed to separate the data (section **??**).

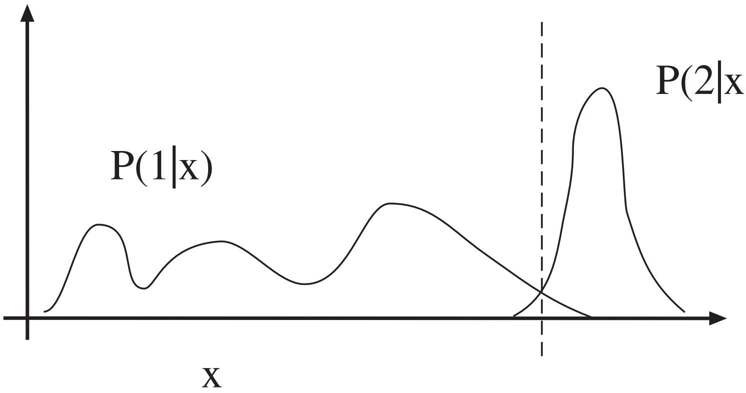


Figure 20.2. The figure shows posterior densities for two classes. The optimal decision boundary is shown as a dashed line. Notice that, while a normal density may provide rather a poor fit *to the posteriors*, the quality of the classifier it provides depends only on *how well it predicts the position of the boundaries.* In this case, assuming that the posteriors are normal may provide a fairly good classifier, because $P(2|x)$ looks normal, and the mean and covariance of $P(1|x)$ look as though they would predict the boundary in the right place.

20.1.3 Example: A Plug-in Classifier for Normal Class-conditional Densities

An important plug-in classifier occurs when the class-conditional densities are known to be normal. We can either assume that the priors are known, or estimate the priors by counting the number of data items from each class. Now we need to provide the parameters for the class-conditional densities. We do this as an estimation problem, using the data items to estimate the mean $\boldsymbol{\mu}_k$ and covariance Σ_k for each class. Now, since $\log a > \log b$ implies $a > b$, we can work with the logarithm of the posterior. This yields a classifier of the form in algorithm 2.

The term $\delta(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k)$ in this algorithm is known as the **Mahalanobis distance** [?]. The algorithm can be interpreted geometrically as saying that the correct class is the one whose mean is closest to the data item, *taking into account the variance.* In particular, distance from a mean along a direction where there

Assume we have N classes, and the k'th class contains N_k examples, of which the i'th is written as $\boldsymbol{x}_{k,i}$.
For each class k, estimate the mean and standard deviation for that class-conditional density.

$$\begin{aligned}\boldsymbol{\mu}_k &= \frac{1}{N_k}\sum_{i=1}^{N_k}\boldsymbol{x}_{k,i}\\ \Sigma_k &= \frac{1}{N_k-1}\sum_{i=1}^{N_k}(\boldsymbol{x}_{k,i}-\boldsymbol{\mu}_k)(\boldsymbol{x}_{k,i}-\boldsymbol{\mu}_k)^T\end{aligned}$$

To classify an example $\boldsymbol{x}$

Choose the class k with the smallest value of $\delta(\mathbf{x};\boldsymbol{\mu}_k,\Sigma_k)^2 - Pr\{k\}$

where

$$\delta(\mathbf{x};\boldsymbol{\mu}_k,\Sigma_k) = \frac{1}{2}\left((\mathbf{x}-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right)^{(1/2)}$$

Algorithm 20.2: *A plug-in classifier can be used to classify objects into classes if the class-conditional densities are known to be normal*

is little variance has a large weight and distance from the mean along a direction where there is a large variance has little weight. This classifier can be simplified by assuming that each class has the same covariance (with the advantage that we have fewer parameters to estimate). In this case, because the term $\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x}$ is common to all expressions, the classifier actually involves comparing expressions that are *linear in $\boldsymbol{x}$* (exercise **??**). If there are only two classes the process boils down to determining whether a linear expression in $\mathbf{x}$ is greater than or less than zero (exercise **??**).

20.1.4 Example: A Non-Parametric Classifier using Nearest Neighbours

It is reasonable to assume that example points "near" an unclassified point should indicate the class of that point. **Nearest neighbours** methods build classifiers using this heuristic. We could classify a point by using the class of the nearest example whose class is known, or use several example points, and make them vote. It is reasonable to require that some minimum number of points vote for the class we choose.

A (k, l) nearest neighbour classifier finds the k example points closest to the point being considered, and classifies this point with the class that has the highest number of votes, as long as this class has more than l votes (otherwise the point is classified as unknown). A $(k, 0)$-nearest neighbour classifier is usually known as a

k**-nearest neighbour classifier**, and a $(1,0)$-nearest neighbour classifier is usually known as a **nearest neighbour classifier**.

Nearest neighbour classifiers are known to be good, in the sense that the risk of using a nearest neighbour classifier with a sufficiently large number of examples lies within quite good bounds of the Bayes risk. As k grows, the difference between the Bayes risk and the risk of using a k-nearest neighbour classifier goes down as $1/\sqrt{k}$; in practice, one seldom uses more than three nearest neighbours. Furthermore, if the Bayes risk is zero, the expected risk of using a k-nearest neighbour classifier is also zero (see [**?**] for more detail on all these points).

Nearest neighbour classifiers come with some computational subtleties, however. The first is the question of finding the k nearest points, which is no mean task in a high-dimensional space. This task can be simplified by noticing that some of the example points may be superfluous. If, when we remove a point from the example set, the set still classifies every point in the space in the same way (the decision boundaries have not moved), then that point is redundant and can be removed. The decision regions for (k,l)-nearest neighbour classifiers are convex polytopes; this makes familiar algorithms available in 2D — where Voronoi diagrams implement the nearest neighbour classifier — but leads to complications in high dimensions, where optimal algorithms are not known as of writing.

```
Given an feature vector $\boldsymbol{x}$
    1. determine the $k$ training examples that are nearest, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$;
    2. determine the class $c$ that has the largest number of representatives $n$ in this set;
    3. if $n > l$, classify $\boldsymbol{x}$ as $c$, otherwise refuse to classify it.
```

Algorithm 20.3: *A (k,l) nearest neighbour classifier uses the type of the nearest training examples to classify a feature vector*

A second difficulty in building such classifiers is the choice of distance. For features that are obviously of the same type, such as lengths, the usual metric may be good enough. But what if one feature is a length, one is a colour, and one is an angle? One possibility is to use a covariance estimate to compute a Mahalanobis-like distance.

20.1.5 Estimating and Improving Performance

Typically, classifiers are chosen to work well on the training set, and this can mean that the performance of the classifier on the training set is a poor guide to its overall performance. One example of this problem is the (silly) classifier that takes any data point and, if it is the same as a point in the training set, emits the class of that point and otherwise chooses randomly between the classes. This classifier has been learnt from data, and has a zero error rate on the training data set; it is likely to be unhelpful on any other data set, though.

The difficulty occurs because classifiers are subject to **overfitting** effects. The phenomenon, which is known by a variety of names (**selection bias** is quite widely used), has to do with the fact that the classifier is chosen to perform well *on the training data set*. The training data is a (possibly representative) subset of the available possibilities. The term overfitting is descriptive of the source of the problem, which is that the classifier's performance on the training data set may have to do with quirks of that data set that don't occur in other sets of examples. If the classifier does this, it is quite possible that it will perform very well on the training data and very badly on any other data set (this phenomenon is often referred to as **generalising badly**).

Generally, we expect classifiers to perform somewhat better on the training set than on the test set (for example, see figure 20.16, which shows training set and test set errors for a classifier that is known to work very well). Overfitting can result in a substantial difference between performance on the training set and performance on the test set. This leaves us with the problem of predicting performance. There are two possible approaches: we can hold back some training data to check the performance of the classifier (an approach we describe below), or we can use theoretical methods to bound the future error rate of the classifier (see, for example, []).

Estimating Total Risk with Cross-Validation

We can make direct estimates of the expected risk of using a classifier, if we split the data set into two subsets, train the classifier on one subset and test it on the other. This is a waste of data, particularly if we have very few data items for a particular class, and may lead to an inferior classifier. However, if the size of the test subset is small, the difficulty may not be significant. In particular, we could then estimate total risk by averaging over all possible splits. This technique, known as **cross-validation**, allows an estimate of the likely future performance of a classifier, at the expense of substantial computation.

The most usual form of this algorithm involves omitting single items from the data set, and is known as **leave-one-out cross-validation**. Errors are usually estimated by simply averaging over the class, but more sophisticated estimates are available [**?**]. We will not justify this tool mathematically; however, it is worth noticing that leave-one-out cross-validation in some sense looks at the sensitivity of the classifier to a small change in the training set. If a classifier performs well under this test, then large subsets of the data set look similar to one another, which suggests that a representation of the relevant probabilities derived from the data set might be quite good.

Using Bootstrapping to Improve Performance

Generally, more training data leads to a better classifier. However, training classifiers with very large data sets can be difficult, and there are diminishing returns.

```
Choose some class of subsets of the training set,
for example, singletons.

For each element of that class, construct a classifier by
omitting that element in training, and compute the
classification errors (or risk) on the omitted subset.

Average these errors over the class of subsets to estimate
the risk of using the classifier trained on the entire training
data set.
```

Algorithm 20.4: *Cross-Validation*

Typically, only a relatively small number of example items are really important in determining the behaviour of a classifier (we see this phenomenon in greater detail in section **??**). The really important examples tend to be rare cases that are quite hard to discriminate — this is because these cases affect the position of the decision boundary most significantly. We need a large data set to ensure that these cases are present, but it appears inefficient to go to great effort to train on a large data set, most of whose elements aren't particularly important.

There is a useful trick that avoids much redundant work. We train on a subset of the examples, run the resulting classifier on the rest of the examples, and then insert the false positives and false negatives into the training set to retrain the classifier. This is because the false positives and false negatives are the cases that give the most information about errors in the configuration of the decision boundaries. This strategy is known as **bootstrapping** (the name is potentially confusing, because there is an unrelated statistical procedure known as bootstrapping; nonetheless, we're stuck with it at this point).

20.2 Building Classifiers from Class Histograms

One simple way to build a probability model for a classifier is to use a histogram. If a histogram is divided by the total number of pixels, we get a representation of the class-conditional probability density function. It is a fact that, as the data set gets larger and the histogram bins get smaller, the histogram divided by the total number of data items will almost certainly converge to the probability density function []. In low dimensional problems, this approach can work quite well (section 20.2.1). It isn't practical for high dimensional data because the number of histogram bins required quickly becomes intractable, unless we use strong independence assumptions to control the complexity (section 20.5).

20.2.1 Finding Skin Pixels using a Classifier

As we indicated in section **??**, skin-finding is useful for activities like building gesture based interfaces. Skin has a quite characteristic range of colours, suggesting that we can build a skin finder by classifying pixels on their colour. Jones and Rehg construct a histogram of RGB values due to skin pixels, and a second histogram of RGB values due to non-skin pixels. These histograms serve as models of the class-conditional densities.

We write $\boldsymbol{x}$ for a vector containing the colour values at a pixel. We subdivide this colour space into boxes, and count the percentage of skin pixels that fall into each box — this histogram supplies $p(\boldsymbol{x}|\text{skin pixel})$, which we can evaluate by determining the box corresponding to $\boldsymbol{x}$ and then reporting the percentage of skin pixels in this box. Similarly, a count of the percentage of non-skin pixels that fall into each box supplies $p(\boldsymbol{x}|\text{not skin pixel})$. We need $p(\text{skin pixel})$ and $p(\text{not skin pixel})$ — or rather, we need only one of the two, as they sum to one. Assume for the moment that the prior is known. We can now build a classifier, using Bayes' rule to obtain the posterior (keep in mind that $p(\boldsymbol{x})$ is easily computed as $p(\boldsymbol{x}|\text{skin pixel}) + p(\boldsymbol{x}|\text{not skin pixel})$).

One way to estimate the prior is to model $p(\text{skin pixel})$ as the fraction of skin pixels in some (ideally large) training set. Notice that our classifier compares

$$\frac{p(\boldsymbol{x}|\text{skin})p(\text{skin})}{p(\boldsymbol{x})}L(\text{skin} \rightarrow \text{not skin})$$

with

$$\frac{p(\boldsymbol{x}|\text{not skin})p(\text{not skin})}{p(\boldsymbol{x})}L(\text{not skin} \rightarrow \text{skin})$$

Now by rearranging terms and noticing that $p(\text{skin}|\boldsymbol{x}) = 1 - p(\text{not skin}|\boldsymbol{x})$, our classifier becomes

- if $p(\text{skin}|\boldsymbol{x}) > \theta$, classify as skin
- if $p(\text{skin}|\boldsymbol{x}) < \theta$, classify as not skin
- if $p(\text{skin}|\boldsymbol{x}) = \theta$, choose classes uniformly and at random

where θ is an expression that doesn't depend on $\boldsymbol{x}$, and encapsulates the relative loss, etc. This yields a family of classifiers, one for each choice of θ.

Each classifier in this family has a different false-positive and false-negative rate. These rates are functions of θ, so we can plot a parametric curve that captures the performance of the family of classifiers. This curve is known as a **receiver operating curve** (or **ROC** for short). Figure 20.4 shows the ROC for a skin finder built using this approach. The ROC is invariant to choice of prior (exercises)— this means that if we change the value of $p(\text{skin})$, we can choose some new value of θ to get a classifier with the same performance. This yields another approach to estimating a prior. We choose some value rather arbitrarily, plot the loss on the training set as a function of θ, and then select the value of θ that minimizes this loss.

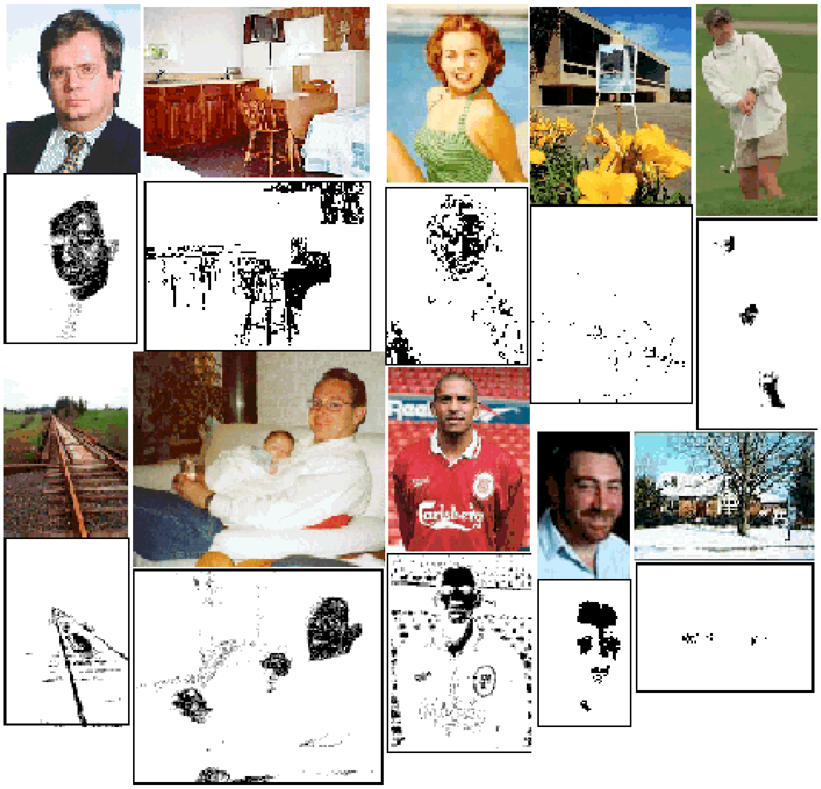


Figure 20.3. The figure shows a variety of images together with the output of the skin detector of Jones and Rehg applied to the image. Pixels marked black are skin pixels, and white are background. Notice that this process is relatively effective, and could certainly be used to focus attention on, say, faces and hands. *figure from Jones and Rehg, Statistical color models with application to skin detection, p.10, in the fervent hope of receiving permission*

20.2.2 Face Finding Assuming Independent Template Responses

Histogram models become impractical in high dimensions, because the number of boxes required goes up as a power of the dimension. We can dodge this phenomenon. Recall from chapter **??** that independence assumptions reduce the number of parameters that must be learned in a probabilistic model; by assuming that terms are independent, we can reduce the dimension sufficiently to use histograms. While this appears to be an aggressive oversimplification, it can result in useful systems. In one such system, due to Schneiderman and Kanade, this model is used to find faces. Assume that the face occurs at a fixed, known scale (we could search smoothed and resampled versions of the image to find larger faces) and will occupy a region of

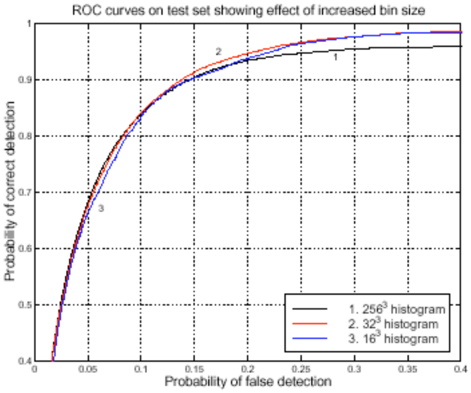


Figure 20.4. The receiver operating curve for the skin detector of Jones and Rehg. This plots the detection rate against the false negative rate for a variety of values of the parameter θ. A perfect classifier has an ROC that, on these axes, is a horizontal line at 100% detection. Notice that the ROC varies slightly with the number of boxes in the histogram. *figure from Jones and Rehg, Statistical color models with application to skin detection, p.11, in the fervent hope of receiving permission*

known shape. In the case of frontal faces, this might be an oval or a square; for a lateral face, this might be some more complicated polygon.

We now need to model the image pattern generated by the face. This is a likelihood model — we want a model giving $P(\text{image pattern}|\text{face})$. As usual, it is helpful to think in terms of generative models — the process by which a face gives rise to an image patch. The set of possible image patches is somewhat difficult to deal with, because it is big, but we can avoid this by dividing the image patch into a set of subregions, and then labelling the subregions, using a small set of labels.

An appropriate labelling can be obtained using a clustering algorithm and a large number of example images. For example, we might cluster the subregions in a large number of example images using k-means; now each cluster center represents a "typical" form of subregion. The subregions in our image patch can then be labelled with the cluster center to which they are closest. This approach has the advantage that minor variations in the image pattern — caused perhaps by noise, or by skin irregularities, etc. — are suppressed.

At this point, a number of models are available. The simplest practical model is to assume that the probability of encountering each pattern is independent of the configuration of the other patterns (but not of position) given that a face is present. This means that our model is:

$$\begin{aligned} P(\text{image}|\text{face}) &= P(\text{label 1 at } (x_1, y_1), \ldots, \text{label } k \text{ at } (x_k, y_k)|\text{face}) \\ &= P(\text{label 1 at } (x_1, y_1)|\text{face}) \ldots P(\text{label } k \text{ at } (x_k, y_k)|\text{face}) \end{aligned}$$

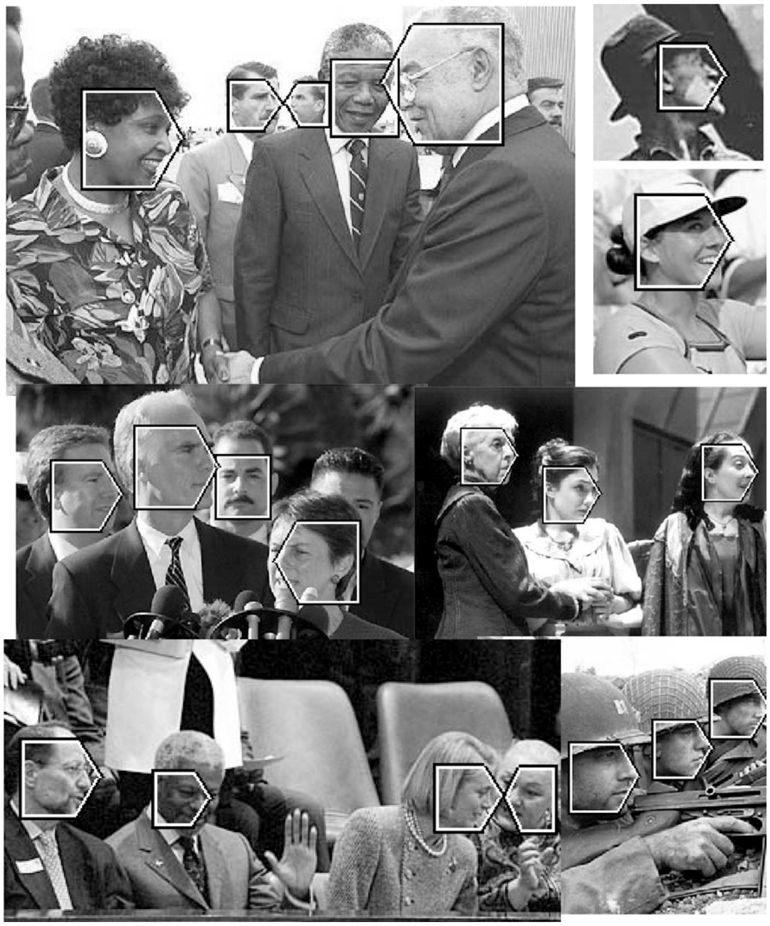


Figure 20.5. Faces found using the method of section 20.5. Image windows at various scales are classified as frontal face, lateral face or non-face, using a likelihood model learned from data. Subregions in the image window are classified into a set of classes learned from data; the face model assumes that labels from these classes are emitted independently of one another, at different positions. This likelihood model yields a posterior value for each class and for each window, and the posterior value is used to identify the window. *figure from Schneiderman and Kanade, A Statistical Method for 3D Object Detection Applied to Faces and Cars, p.6, in the fervent hope of receiving permission*

In this case, each term of the form $P(\text{label } k \text{ at } (x_k, y_k)|\text{face})$ can be learned fairly easily by labelling a large number of example images, and then forming a histogram. Because the histograms are now two dimensional, the number of boxes is no longer problematic. A similar line of reasoning leads to a model of $P(\text{image}|\text{no face})$. A classifier follows from the line of reasoning given above. This approach has been used successfully by Schneiderman and Kanade to build detectors for faces and cars (figure 20.5).

20.3 Feature Selection

Assume we have a set of pixels that we believe belong together, and that should be classified. What features should we present to a classifier? One approach is to present all the pixel values: this gives the classifier the maximum possible amount of information about the set of pixels, but creates a variety of problems.

Firstly, high dimensional spaces are "big" in the sense that very large numbers of examples can be required to represent the available possibilities fairly. For example, a face at low resolution has a fairly simple structure: it consists (rather roughly) of some dark bars (the eyebrows and eyes) and light bars (the specular reflections from the nose and forehead) on a textureless background. However, if we are working with high resolution faces, it might be very difficult to supply enough examples to make determine that this structure is significant and that minor variations in skin texture, etc. are irrelevant. Instead, we would like to choose a feature space that would make these properties "obvious", typically by imposing some form of structure on the examples.

Secondly, we may know some properties of the patterns in advance; for example, we have models of the behaviour of illumination. Forcing a classifier to use examples to, in essence, come up with a model that we already know is a waste of examples. We would like to use features that are consistent with our knowledge of the patterns. This might involve preprocessing regions (for example, to remove the effects of illumination changes), or choosing features that are invariant to some kinds of transformation (for example, scaling an image region to a standard size).

You should notice a similarity between feature selection and model selection (as described in sections **??** and **??**). In model selection, we were attempting to obtain a model that best explains a data set; here we are attempting to a set of features that best classifies a data set. The two are basically the same activity in slightly distinct forms (you can view a set of features as a model, and classification as explanation); here we will describe methods that are used mainly for feature selection. We concentrate on two standard methods for obtaining **linear features**, features which are a linear function of the initial feature set.

20.3.1 Principal Component Analysis

The core goal in feature selection is to obtain a smaller set of features that "accurately represents" the original set. What this means rather depends on the application; however, one important possibility is that the new set of features should capture as much of the old set's variance as possible. The easiest way to see this is to consider an extreme example; if the value of one feature can be predicted precisely from the value of the others, then it is clearly redundant and can be dropped. By this argument, if we are going to drop a feature, the best one to drop is the one whose value is most accurately predicted by the others. We can do more than drop features: we can make new features as functions of the old features.

In **principal component analysis**, the new features are linear functions of the old features. In principal component analysis, we take a set of data points and construct a lower dimensional linear subspace that "best explains" the variation of these data points from their mean. This method (also known as the Karhunen-Loéve transform) is a classical technique from statistical pattern recognition [**?**; **?**; **?**].

Assume we have a set of n feature vectors $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ in $\mathbb{R}^d$. The mean of this set of feature vectors is $\boldsymbol{\mu}$ (you should think of the mean as the center of gravity in this case), and their covariance is Σ (you can think of the variance as a matrix of second moments). We use the mean as an origin, and study the offsets from the mean, $(\boldsymbol{x}_i - \boldsymbol{\mu})$.

Our features will be linear combinations of the original features; this means it is natural to consider the projection of these offsets onto various different directions. A unit vector $\boldsymbol{v}$ represents a direction in the original feature space; we can interpret this direction as a new feature $v(\boldsymbol{x})$. The value of u on the i'th data point is given by $v(\boldsymbol{x}_i) = \boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu})$. A good feature will capture as much of the variance of the original data set as possible. Notice that v has zero mean; then the variance of v is

$$\begin{aligned}
\mathrm{var}(v) &= \frac{1}{n-1}\sum_{i=1}^{n} v(\boldsymbol{x}_i)v(\boldsymbol{x}_i)^T \\
&= \frac{1}{n}\sum_{i=1}^{n-1} \boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{v}^T(\boldsymbol{x}_i - \boldsymbol{\mu}))^T \\
&= \boldsymbol{v}^T \left\{ \sum_{i=1}^{n-1} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T \right\} \boldsymbol{v} \\
&= \boldsymbol{v}^T \Sigma \boldsymbol{v}
\end{aligned}$$

Now we should like to maximise $\boldsymbol{v}^T\Sigma\boldsymbol{v}$ subject to the constraint that $\boldsymbol{v}^T\boldsymbol{v} = 1$. This is an eigenvalue problem; the eigenvector of Σ corresponding to the largest eigenvalue is the solution. Now if we were to project the data onto a space *perpendicular* to this eigenvector, we would obtain a collection of $d - 1$ dimensional vectors. The highest variance feature for this collection would be the eigenvector of Σ with second largest eigenvalue; and so on.

This means that the eigenvectors of Σ — which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue — give a set of features with the following properties:

- They are independent (because the eigenvectors are orthogonal).
- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that preserves the most variance.

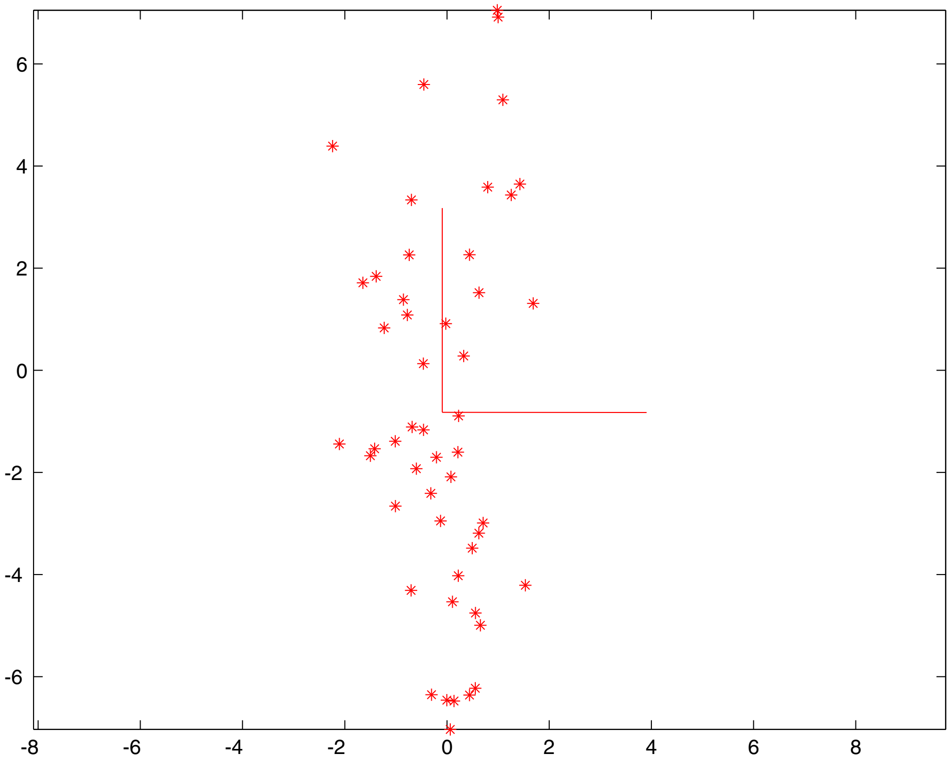


Figure 20.6. A data set which is well represented by a principal component analysis. The axes represent the directions obtained using PCA; the vertical axis is the first principal component, and is the direction in which the variance is highest.

You should notice that, depending on the data source, principal components can give a very good or a very bad representation of a data set (see figures 20.6 and 20.7, and figure 20.8).

20.3.2 Canonical Variates

Principal component analysis yields a set of linear features of a particular dimension that best represents the variance in a high-dimensional dataset. There is no guarantee that this set of features is good for *classification*. For example, figure 20.8 shows a dataset where the first principal component would yield a very bad classifier and the second principal component would yield quite a good one, despite not capturing the variance of the data set.

Linear features that emphasize the distinction between classes are known as **canonical variates**. To construct canonical variates, assume that we have a set of

Assume we have a set of n feature vectors $\boldsymbol{x}_i$ $(i = 1, \ldots, n)$ in $\mathbb{R}^d$. Write

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{n} \sum_i \boldsymbol{x}_i \\ \Sigma &= \frac{1}{n-1} \sum_i (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T \end{aligned}$$

The unit eigenvectors of Σ — which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue — give a set of features with the following properties:

- They are independent.
- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that preserves the most variance.

Algorithm 20.5: *Principal components analysis identifies a collection of linear features that are independent, and capture as much variance as possible from a dataset.*

data items $\boldsymbol{x}_i$, for $i \in \{1, \ldots, n\}$. We assume that there are p features (i.e. that the $\boldsymbol{x}_i$ are p-dimensional vectors). We have g different classes, and the j'th class has mean $\boldsymbol{\mu}_j$. Write $\overline{\boldsymbol{\mu}}$ for the mean of the class means, i.e.

$$\overline{\boldsymbol{\mu}} = \frac{1}{g} \sum_{j=1}^{g} \boldsymbol{\mu}_j$$

Write

$$\mathcal{B} = \frac{1}{g-1} \sum_{j=1}^{g} (\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})(\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})^T$$

Note that $\mathcal{B}$ gives the variance of the class means. In the simplest case, we assume that each class has the same covariance Σ, and that this has full rank. We would like to obtain a set of axes where the clusters of data points belonging to a particular class will group together tightly, while the distinct classes will be widely separated. This involves finding a set of features that maximises the ratio of the separation (variance) between the class means to the variance within each class. The separation between the class means is typically referred to as the **between-class variance**, and the variance within a class is typically referred to as the **within-class variance**.

Now we are interested in linear functions of the features, so we concentrate on

$$v(\boldsymbol{x}) = \boldsymbol{v}^T \boldsymbol{x}$$

We should like to maximize the ratio of the between-class variances to the within-class variances for $\boldsymbol{v}_1$.

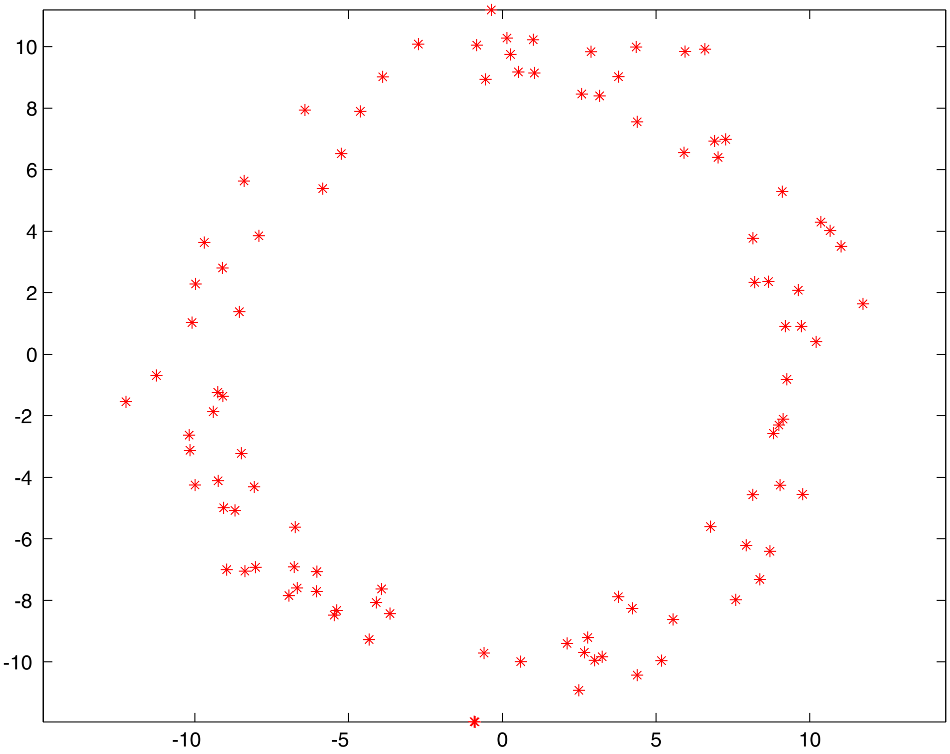


Figure 20.7. Not every data set that is well represented by PCA. The principal components of this data set will be relatively unstable, because the variance in each direction is the same for the source. This means that we may well report significantly different principal components for different datasets from this source. This is a secondary issue — the main difficulty is that projecting the data set onto some axis will suppress the main feature, its circular structure.

Using the same argument as for principal components, we can achieve this by choosing $\boldsymbol{v}$ to maximise

$$\frac{\boldsymbol{v}_1^T \mathcal{B} \boldsymbol{v}_1}{\boldsymbol{v}_1^T \Sigma \boldsymbol{v}_1}$$

This problem is the same as maximising $\boldsymbol{v}_1^T \mathcal{B} \boldsymbol{v}_1$ subject to the constraint that $\boldsymbol{v}_1^T \Sigma \boldsymbol{v}_1 = 1$. In turn, a solution has the property that

$$\mathcal{B}\boldsymbol{v}_1 + \lambda \Sigma \boldsymbol{v}_1 = 0$$

for some constant λ. This is known as a **generalised eigenvalue problem** — if Σ has full rank, we can solve it by finding the eigenvector of $\Sigma^{-1}\mathcal{B}$ with largest

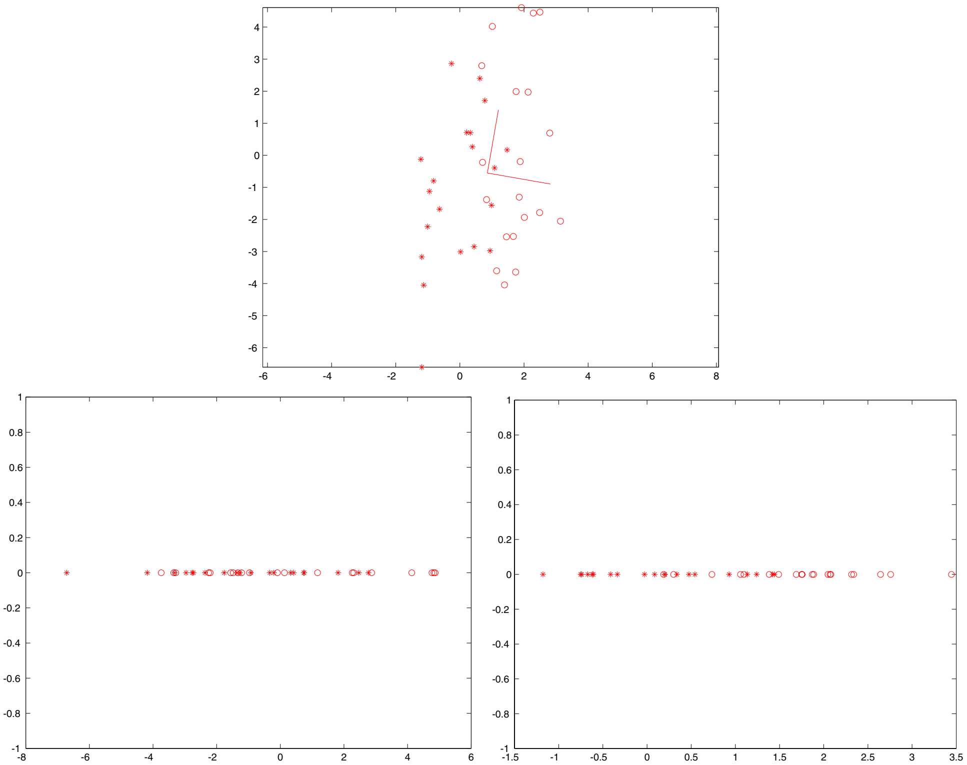


Figure 20.8. Principal component analysis doesn't take into account the fact that there may be more than one class of item in a dataset. This can lead to significant problems. For a classifier, we would like to obtain a set of features that firstly reduces the number of features and secondly makes the difference between classes most obvious. For the data set on the **top**, one class is indicated by circles and the other by stars. PCA would suggest projection onto a vertical axis, which captures the variance in the dataset, but cannot be used to discriminate it, as we can see from the axes obtained by PCA, which are overlaid on the data set. The **bottom row** shows the projections onto those axes. On the **bottom left**, we show the projection onto the first principal component — which has higher variance, but separates the classes poorly — and on the **bottom right**, we show the projection onto the second principal component — which has significantly lower variance (look at the axes) and gives better separation.

eigenvalue (otherwise, we use specialised routines within the relevant numerical software environment).

Now for each $\boldsymbol{v}_l$, for $2 \leq l \leq p$, we should like to find features that extremise the criterion, and are independent of the the previous $\boldsymbol{v}_l$. These are provided by the other eigenvectors of $\Sigma\mathcal{B}$. The eigenvalues give the variance along the features (which are independent). By choosing the $m < p$ eigenvectors with the largest

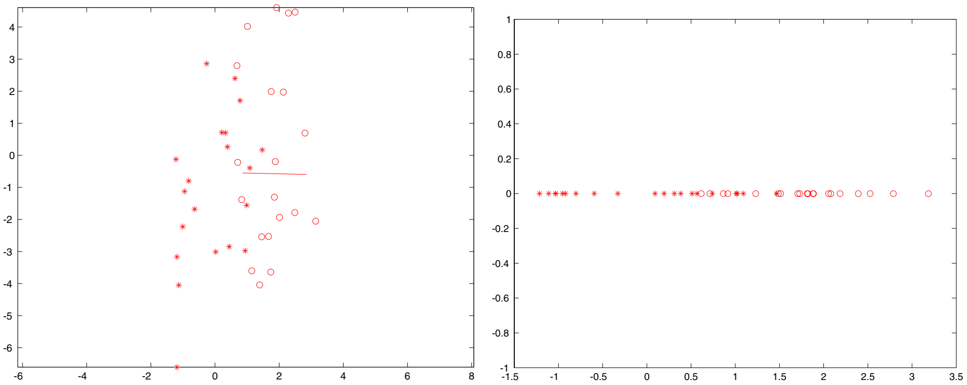


Figure 20.9. Canonical variates use the class of each data item as well as the features in estimating a good set of linear features. In particular, the approach constructs axes that separate different classes as well as possible. The data set used in figure 20.8 is shown on the **left**, with the axis given by the first canonical variate overlaid. On the **bottom right**, we show the projection onto that axis, where the classes are rather well separated.

eigenvalues, we obtain a set of features that reduces the dimension of the feature space will best preserving the separation between classes. This doesn't by any means guarantee the best error rate for a classifier on a reduced number of features, but it offers a good place to start, by reducing the number of features while respecting the category structure (details and examples in [**?**], p— or in [**?**], p—).

If the classes don't have the same covariance, it is still possible to construct canonical variates. In this case, we estimate a Σ as the covariance of all of the offsets of each data item *from its own class mean*, and proceed as before. Again, this is an approach without a guarantee of optimality, but one that can work quite well in practice.

20.4 Neural Networks

It is commonly the case that neither simple parametric density models nor histogram models can be used. In this case, we must either use more sophisticated density models (an idea we explore in this section) or look for decision boundaries directly (section **??**).

20.4.1 Key Ideas

A **neural network** is a parametric approximation technique that has proven useful for building density models. Neural networks typically approximate a vector function $\boldsymbol{f}$ of some input $\boldsymbol{x}$ with a series of **layers**. Each layer forms a vector of outputs each of which is obtained by applying the same non-linear function — which we shall write as ϕ — to different affine functions of the inputs. We adopt the con-

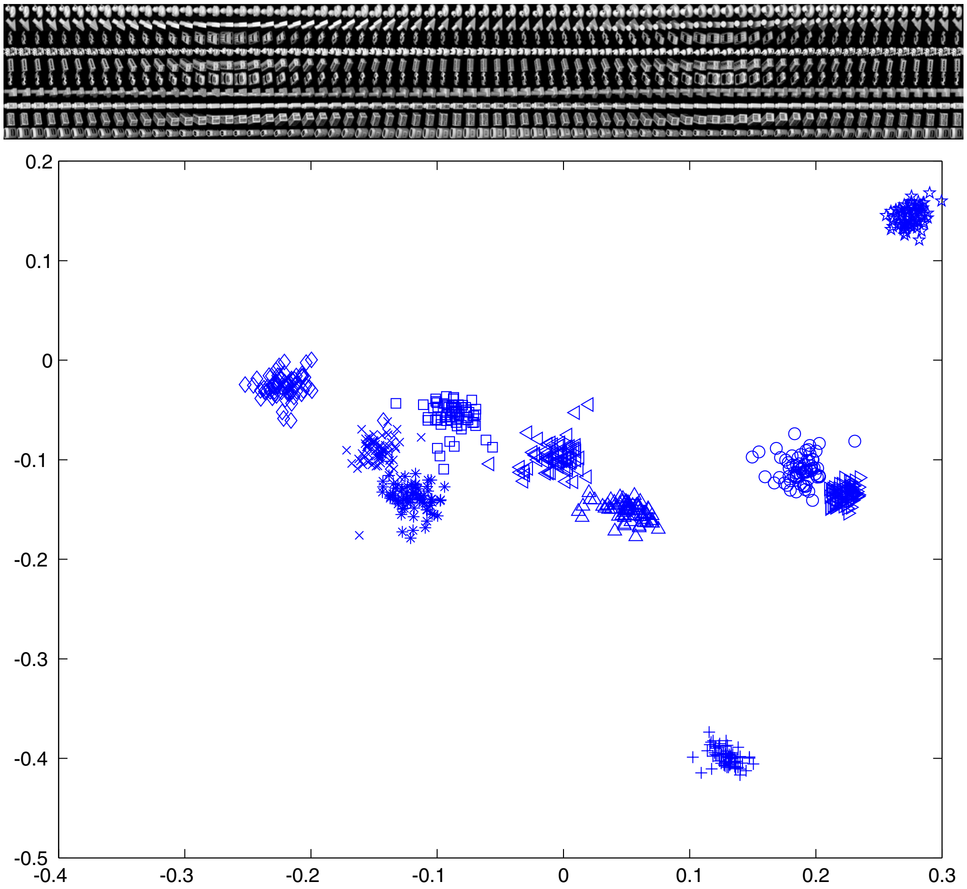


Figure 20.10. Canonical variates are effective for a variety of simple template matching problems. The figure on top shows views of 10 objects at a variety of poses, on a black background (these images are smoothed and resampled versions of images in the well-known COIL database, due to Nene and Nayar and available at `http://www.cs.columbia.edu/*******`). Identifying an object from one of these images is a relatively simple matter, because there is no segmentation. We then used 60 of the images of each object to determine a set of canonical variates. The figure below shows the first two canonical variates for 71 images — the 60 training images and 11 others — of each object (different symbols correspond to different objects). Note that the clusters are tight and well separated; on these two canonical variates alone, we could probably get quite good classification.

venient trick of adding an extra component to the inputs and fixing the value of this component at one, so that we obtain a linear function of this augmented input vector. This means that a layer with augmented input vector $\boldsymbol{u}$ and output vector

Assume that we have a set of data items of g different classes. There are n_k items in each class, and a data item from the k'th class is $\boldsymbol{x}_{k,i}$, for $i \in \{1, \ldots, n_k\}$. The j'th class has mean $\boldsymbol{\mu}_j$. We assume that there are p features (i.e. that the $\boldsymbol{x}_i$ are p-dimensional vectors).
Write $\overline{\boldsymbol{\mu}}$ for the mean of the class means, i.e.

$$\overline{\boldsymbol{\mu}} = \frac{1}{g}\sum_{j=1}^{g}\boldsymbol{\mu}_j$$

Write

$$\mathcal{B} = \frac{1}{g-1}\sum_{j=1}^{g}(\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})(\boldsymbol{\mu}_j - \overline{\boldsymbol{\mu}})^T$$

Assume that each class has the same covariance Σ, which is either known or estimated as

$$\Sigma = \frac{1}{N-1}\sum_{c=1}^{g}\left\{\sum_{i=1}^{n_c}(\boldsymbol{x}_{c,i} - \boldsymbol{\mu}_c)(\boldsymbol{x}_{c,i} - \boldsymbol{\mu}_c)^T\right\}$$

The unit eigenvectors of $\Sigma^{-1}\mathcal{B}$ — which we write as $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d$, where the order is given by the size of the eigenvalue and $\boldsymbol{v}_1$ has the largest eigenvalue — give a set of features with the following property:

- Projection onto the basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ gives the k-dimensional set of linear features that best separates the class means.

Algorithm 20.6: *Canonical variates identifies a collection of linear features that separating the classes as well as possible.*

$\boldsymbol{v}$ can be written as

$$\boldsymbol{v} = [\phi(\boldsymbol{w}_1 \cdot \boldsymbol{u}), \phi(\boldsymbol{w}_2 \cdot \boldsymbol{u}), \ldots \phi(\boldsymbol{w}_n \cdot \boldsymbol{u})]$$

where the $\boldsymbol{w}_i$ are parameters that can be adjusted to approve the approximation.

Typically, a neural net uses a sequence of layers to approximate a function. Each layer will use augmented input vectors. For example, if we are approximating a vector function $\boldsymbol{g}$ of a vector $\boldsymbol{x}$ with a two layer net, we obtain

$$\boldsymbol{g}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x}) = [\phi(\boldsymbol{w}_{21} \cdot \boldsymbol{y}), \phi(\boldsymbol{w}_{22} \cdot \boldsymbol{y}), \ldots \phi(\boldsymbol{w}_{2n} \cdot \boldsymbol{y})]$$

where

$$\boldsymbol{y}(\boldsymbol{z}) = [\phi(\boldsymbol{w}_{11} \cdot \boldsymbol{z}), \phi(\boldsymbol{w}_{12} \cdot \boldsymbol{z}), \ldots \phi(\boldsymbol{w}_{1m} \cdot \boldsymbol{z}), 1]$$

and

$$\boldsymbol{z}(\boldsymbol{x}) = [x_1, x_2, \ldots, x_p, 1]$$

Some of the elements of $\boldsymbol{w}_{1k}$ or $\boldsymbol{w}_{2k}$ could be clamped at zero; in this case, we are insisting that some elements of $\boldsymbol{y}$ do not affect $\boldsymbol{f}(\boldsymbol{x})$. If this is the case, the layer is referred to as being **partially connected**, otherwise it is known as a **fully connected layer**. Of course, layer two could be either fully or partially connected as well. The parameter n is fixed by the dimension of $\boldsymbol{f}$ and p is fixed by the dimension of $\boldsymbol{x}$, but there is no reason that m should be the same as either n or p. Typically, m is larger than either. A similar construction yields three layer networks (or networks with more layers, which are uncommon). Neural networks are often drawn with circles indicating variables and arrows indicating possibly non-zero connections; this gives a representation that exposes the basic structure of the approximation (figure 20.11).

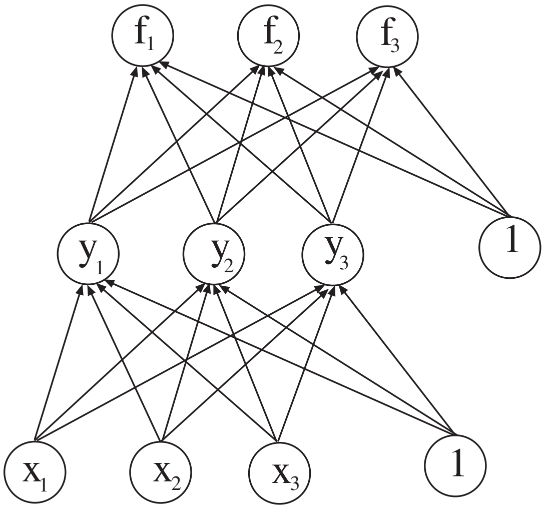


Figure 20.11. Neural networks are often illustrated by diagrams of the form shown above. Each circle represents a variable, and the circles are typically labelled with the variable. The "layers" are obvious in such a drawing. This network is the two layer network given in the text; the arrows indicate that the coefficient coupling the two variables in the affine function could be non-zero. This network is fully connected, because all arrows are present. It is possible to have arrows skip layers, etc.

Choosing a Non-Linearity

There are a variety of possibilities for ϕ. For example, we could use a **threshold function**, which has value one when the argument is positive and zero otherwise. It is quite easy to visualize the response of a layer of that uses a threshold function; each component of the layer changes from zero to one along a hyperplane. This means that the output vector takes different values in each cell of an arrangement of hyperplanes in the input space. Networks that use layers of this form are hard to train, because the threshold function is not differentiable.

It is more common to use a ϕ that changes smoothly (but rather quickly) from zero to one, often called a **sigmoid function** or **squashing function**. The **logistic function** is one popular example. This is a function of the form

$$\phi(x;\nu) = \frac{e^{x/\nu}}{1+e^{x/\nu}}$$

where ν controls how sharply the function changes at $x = 0$. It isn't crucial that the horizontal assymptotes are zero and one. Another popular squashing function is

$$\phi(x;\nu,A) = A\tanh(\nu x)$$

which has horizontal assymptotes at A and $-A$. Figure 20.12 illustrates these non-linearities.

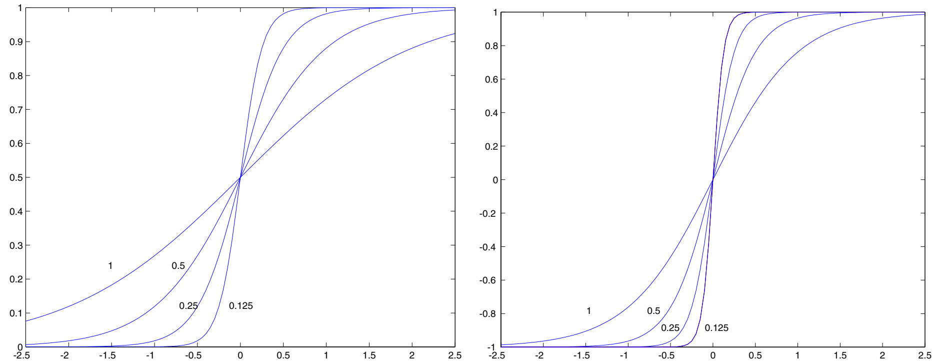


Figure 20.12. On the **left**, a series of squashing functions obtained using $\phi(x;\nu) = \frac{e^{x/\nu}}{1+e^{x/\nu}}$, for different values of ν indicated on the figure. On the **right**, a series of squashing functions obtained using $\phi(x;\nu,A) = A\tanh(x/\nu)$ for different values of ν indicated on the figure. Generally, for x close to the center of the range, the squashing function is linear; for x small or large, it is strongly non-linear.

Producing a Classifier using a Neural Net

To produce a neural net that approximates some function $\boldsymbol{g}(\boldsymbol{x})$, we collect a series of examples $\boldsymbol{x}^e$. We construct a network that has one output for each dimension of $\boldsymbol{g}$. Write this network as $\boldsymbol{n}(\boldsymbol{x};\boldsymbol{p})$, where $\boldsymbol{p}$ is a parameter vector that contains all the $\boldsymbol{w}_{ij}$. We supply a desired output vector $\boldsymbol{o}^e$ for this input; typically, $\boldsymbol{o}^e = \boldsymbol{g}(\boldsymbol{x}^e)$. We now obtain $\hat{\boldsymbol{p}}$ that minimizes

$$Error(\boldsymbol{p}) = \left(\frac{1}{2}\right)\sum_e \mid \boldsymbol{n}(\boldsymbol{x}^e;\boldsymbol{p}) - \boldsymbol{o}^e \mid^2$$

using appropriate optimization software (the half will simplify a little notation later on, but is of no real consequence).

The most significant case occurs when $\boldsymbol{g}(\boldsymbol{x})$ is intended to approximate the posterior on classes, given the data. We do not know this posterior, and so cannot supply its value to the training procedure. Instead, we require that our network has one output for each class. Given an example $\boldsymbol{x}^e$, we construct a desired output $\boldsymbol{o}^e$ as a vector which contains a one in the component corresponding to that example's class and a zero in each other component. We now train the net as above, and regard the output of the neural network as a model of the posterior probability. An input $\boldsymbol{x}$ is then classified by forming $\boldsymbol{n}(\boldsymbol{x}; \hat{\boldsymbol{p}})$, and then choosing the class that corresponds to the largest component of this vector.

20.4.2 Minimizing the Error

Recall that we are training a net by minimizing the sum over the examples of the difference between the desired output and the actual output, i.e., by minimizing

$$Error(\boldsymbol{p}) = \left(\frac{1}{2}\right) \sum_e \mid \boldsymbol{n}(\boldsymbol{x}^e; \boldsymbol{p}) - \boldsymbol{o}^e \mid^2$$

as a function of the parameters $\boldsymbol{p}$. There are a variety of strategies for obtaining $\hat{\boldsymbol{p}}$, the set of parameters that minimize this error. One is gradient descent; from some initial point $\boldsymbol{p}_i$, we compute a new point $\boldsymbol{p}_{i+1}$ by

$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \epsilon(\nabla Error)$$

where ϵ is some small constant.

Stochastic Gradient Descent

Write the error for example e as $Error(\boldsymbol{p}; \boldsymbol{x}^e)$, so the total error is $Error(\boldsymbol{p}) = \sum_e Error(\boldsymbol{p}; \boldsymbol{x}^e)$. Now if we use gradient descent, we are updating parameters using the algorithm

$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \epsilon \nabla Error$$

(where the gradient is with respect to $\boldsymbol{p}$ and is evaluated at $\boldsymbol{p}_i$). This works, because if ϵ is sufficiently small, we have that

$$\begin{aligned} Error(\boldsymbol{p}_{i+1}) &= Error(\boldsymbol{p}_i - \epsilon \nabla Error) \\ &\approx Error(\boldsymbol{p}_i) - \epsilon(\nabla Error \cdot \nabla Error) \\ &\leq Error(\boldsymbol{p}_i) \end{aligned}$$

with equality only at an extremum. This creates a problem: evaluating the error and its gradient is going to involve a sum over all examples, which may be a very large number. We should like to avoid this sum; it turns out that it is possible to do so by selecting an example at random, computing the gradient *for that example* *alone*, and updating the parameters using that gradient. In this process, known as **stochastic gradient descent**, we update the parameters using the algorithm

$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \epsilon \nabla Error(\boldsymbol{p}; \boldsymbol{x}^e)$$

(where the gradient is with respect to $\boldsymbol{p}$, is evaluated at $\boldsymbol{p}_i$, and we choose the example uniformly at random, making a different choice at each step). In this case, the error doesn't necessarily go down for each particular choice, but the *expected* value of the error *does* go down, for a sufficiently small value of ϵ. In particular, we have

$$\begin{aligned}
\mathrm{E}(Error(\boldsymbol{p}_{i+1})) &= \mathrm{E}(Error(\boldsymbol{p}_i - \epsilon \nabla Error(\boldsymbol{p}; \boldsymbol{x}^e)) \\
&\approx \mathrm{E}(Error(\boldsymbol{p}_i) - \epsilon(\nabla Error \cdot \nabla Error(\boldsymbol{p}; \boldsymbol{x}^e)) \\
&= Error(\boldsymbol{p}_i) - \epsilon \frac{1}{n} \sum_e (\nabla Error \cdot \nabla Error(\boldsymbol{p}; \boldsymbol{x}^e)) \\
&= Error(\boldsymbol{p}_i) - \epsilon(\nabla Error \cdot (\frac{1}{n} \sum_e \nabla Error(\boldsymbol{p}; \boldsymbol{x}^e))) \\
&= Error(\boldsymbol{p}_i) - \frac{\epsilon}{n}(\nabla Error \cdot \nabla Error) \\
&< Error(\boldsymbol{p}_i) \text{ if } |\nabla E| > 0
\end{aligned}$$

By taking sufficient steps down a gradient computed using only one example (selected uniformly and at random each time we take a step), we can in fact minimize the function. This is because the expected value goes down for each step, unless we're at the minimum.

Backpropagation

The difficulty with this problem is that $\nabla Error$ could be quite hard to compute. There is an effective strategy for computing $\nabla Error$ called **back propagation**. This approach exploits the layered structure of the neural network as a function of a function of a ... to obtain the derivative.

Now recall the two layer neural net which we wrote as

$$\boldsymbol{f}(\boldsymbol{x}) = [\phi(\boldsymbol{w}_{21} \cdot \boldsymbol{y}), \phi(\boldsymbol{w}_{22} \cdot \boldsymbol{y}), \ldots \phi(\boldsymbol{w}_{2n} \cdot \boldsymbol{y})]$$

where

$$\boldsymbol{y}(\boldsymbol{z}) = [\phi(\boldsymbol{w}_{11} \cdot \boldsymbol{z}), \phi(\boldsymbol{w}_{12} \cdot \boldsymbol{z}), \ldots \phi(\boldsymbol{w}_{1m} \cdot \boldsymbol{z}), 1]$$

and

$$\boldsymbol{z}(\boldsymbol{x}) = [x_1, x_2, \ldots, x_p, 1]$$

We would like to compute

$$\frac{\partial Error}{\partial w_{kl,m}}$$

```
Choose $\boldsymbol{p}_o$ (randomly)
Use backpropagation (algorithm ??) to compute
      $\nabla Error(\boldsymbol{x}^e; \boldsymbol{p}_o)$
  $\boldsymbol{p}_n = \boldsymbol{p}_o - \epsilon \nabla Error(\boldsymbol{x}^e; \boldsymbol{p}_o)$
Until $|\, Error(\boldsymbol{p}_n) - Error(\boldsymbol{p}_o) \,|$ is small
      or $|\, \boldsymbol{p}_o - \boldsymbol{p}_n \,|$ is small

      $\boldsymbol{p}_o = \boldsymbol{p}_n$
      Choose an example $(\boldsymbol{x}^e, \boldsymbol{o}^e)$ uniformly and
         at random from the training set
      Use backpropagation (algorithm ??) to compute
         $\nabla Error(\boldsymbol{x}^e; \boldsymbol{p}_o)$
      $\boldsymbol{p}_n = \boldsymbol{p}_o - \epsilon \nabla Error(\boldsymbol{x}^e; \boldsymbol{p}_o)$
end
```

Algorithm 20.7: Stochastic gradient descent minimizes the error of a neural net approximation, using backpropagation to compute the derivatives.

where $w_{kl,m}$ is the m'th component of $\boldsymbol{w}_{kl}$. Let us deal with the coefficients of the output layer first, so that we are interested in $w_{2l,m}$, and we get

$$\begin{aligned}
\frac{\partial Error}{\partial w_{2l,m}} &= \sum_k \frac{\partial Error}{\partial f_k} \frac{\partial f_k}{\partial w_{2l,m}} \\
&= \frac{\partial Error}{\partial f_l} \frac{\partial f_l}{\partial w_{2l,m}} \\
&= \sum_e \left\{ (f_l(\boldsymbol{x}^e) - o_l^e) \frac{\partial f_l}{\partial w_{2l,m}} \right\} \\
&= \sum_e \{ (f_l(\boldsymbol{x}^e) - o_l^e) \phi'_{2l}(y_m(\boldsymbol{x}^e)) \} \\
&= \sum_e \{ \delta_{2l}^e (y_m(\boldsymbol{x}^e)) \}
\end{aligned}$$

Here we use the notation

$$\phi'_{2l} = \frac{\partial \phi}{\partial u}$$

where the derivative is evaluated at $u = \boldsymbol{w}_{21} \cdot \boldsymbol{y}$, and we write

$$\delta_{2l}^e = (f_l(\boldsymbol{x}^e) - o_l^e)\phi'_{2l}$$

Notice that evaluating this derivative involves terms in the input of the layer — the terms $y_m(\boldsymbol{x}^e)$ — and in its output — the terms δ_{2l}^e.

Now consider the coefficients of the second layer. We are interested in $w_{1l,m}$, and we get

$$\begin{aligned}
\frac{\partial Error}{\partial w_{1l,m}} &= \sum_k \left\{ \frac{\partial Error}{\partial f_k} \frac{\partial f_k}{\partial w_{1l,m}} \right\} \\
&= \sum_{i,j} \left\{ \frac{\partial Error}{\partial f_i} \frac{\partial f_i}{\partial y_j} \frac{\partial y_j}{\partial w_{1l,m}} \right\} \\
&= \left\{ \sum_k \frac{\partial Error}{\partial f_k} \frac{\partial f_k}{\partial y_l} \right\} \frac{\partial y_l}{\partial w_{1l,m}} \\
&= \sum_e \left\{ \sum_k \left\{ (f_k(\boldsymbol{x}^e) - o_k^e) \frac{\partial f_k}{\partial y_l} \right\} \frac{\partial y_l}{\partial w_{1l,m}} \right\} \\
&= \sum_e \left\{ \sum_k \{(f_k(\boldsymbol{x}^e) - o_k^e)\phi'_{2k} w_{2k,l}\} \frac{\partial y_l}{\partial w_{1l,m}} \right\} \\
&= \sum_e \left\{ \sum_k \{(f_k(\boldsymbol{x}^e) - o_k^e)\phi'_{2k} w_{2k,l}\} \phi'_{1l} z_m \right\} \\
&= \sum_e \left\{ \sum_k \}\delta^e_{2k} w_{2k,l}\} \phi'_{1l} z_m \right\}
\end{aligned}$$

In this expression,

$$\phi'_{2k} = \frac{\partial \phi}{\partial u}$$

evaluated at $u = \boldsymbol{w}_{2k} \cdot \boldsymbol{y}$, and

$$\phi'_{1l} = \frac{\partial \phi}{\partial u}$$

evaluated at $u = \boldsymbol{w}_{1l} \cdot \boldsymbol{z}$. Now if we write

$$\delta^e_{1l} = \sum_k \{\delta^e_{2k} w_{2k,l}\} \phi'_{1l}$$

we get

$$\frac{\partial E}{\partial w_{1l,m}} = \sum_e \delta^e_{1l} z_m(\boldsymbol{x}^e)$$

Again, this sum involves a term obtained computing the previous derivative, terms in the derivatives within the layer, and terms in the input. You should convince yourself that, if we had a third layer, the derivative of the error with respect to parameters within this third layer would have a similar form — a function of terms in the derivative of the second layer, terms in the derivatives within the third layer, and terms in the input (all this comes from aggressive application of the chain rule). This suggests a two pass algorithm:

1. Evaluate the net's output on each example. This is usually referred to as a **forward pass**.
2. Evaluate the derivatives using the intermediate terms. This is usually referred to as a **backward pass**.

This process yields the derivatives of the total error with respect to the parameters. We can obtain another simplification: we adopted stochastic gradient descent to avoid having to sum the value of the error and of its gradient over all examples. Because computing a gradient is linear, to compute the gradient of the error on one example alone, we simply drop the sum at the front of our expressions for the gradient. The whole is given in algorithm **??**.

20.4.3 When to Stop Training

Typically, gradient descent is not continued until an exact minimum is found. Surprisingly, this is a source of robustness. The easiest way to understand this is to consider the shape of the error function around the minimum. If the error function changes sharply at the minimum, then the performance of the network is quite sensitive to the choice of parameters. This suggests that the network will generalize badly. You can see this by assuming that the training examples are one half of a larger set; if we had trained the net on the other half, we'd have obtained slightly different set of parameters. This means that the net with our current parameters will perform badly on this other half, because the error changes sharply with a small change in the parameters.

Now if the error function doesn't change sharply at the minimum, there is no particular point in expending effort to be at the minimum value, as long as we are reasonably close — we know that this minimum error value won't be attained on a training set. It is common practice to continue with stochastic gradient descent until (a) each example will have been visited on average rather more than once and (b) the decrease in the value of the function goes below some threshold.

A more difficult question is how many layers to use, and how many units to use in each layer. This question — which is one of model selection — tends to be resolved by experiment. We refer interested readers to [].

20.4.4 Finding Faces using Neural Networks

Face finding is an application that illustrates the usefulness of classifiers. In frontal views at a fairly coarse scale, all faces look basically the same; there are bright regions on the forehead, the cheeks and the nose, and dark regions around the eyes, the eyebrows, the base of the nose and the mouth. This suggests approaching face finding as a search over all image windows of a fixed size for windows that look like a face. Larger or smaller faces can be found by searching coarser or finer scale images.

Because a face illuminated from the left looks very different to a face illuminated from the right, the image windows must be corrected for illumination. Generally,

Notation:
Write the two-layer neural net as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x};\boldsymbol{p}) &= [\phi(\boldsymbol{w}_{21}\cdot\boldsymbol{y}),\phi(\boldsymbol{w}_{22}\cdot\boldsymbol{y}),\ldots\phi(\boldsymbol{w}_{2n}\cdot\boldsymbol{y})] \\ \boldsymbol{y}(\boldsymbol{z}) &= [\phi(\boldsymbol{w}_{11}\cdot\boldsymbol{z}),\phi(\boldsymbol{w}_{12}\cdot\boldsymbol{z}),\ldots\phi(\boldsymbol{w}_{1m}\cdot\boldsymbol{z}),1] \\ \boldsymbol{z}(\boldsymbol{x}) &= [x_1,x_2,\ldots,x_p,1] \end{aligned}$$

($\boldsymbol{p}$ is a vector containing all parameters). Write the error on a single example as

$$\begin{aligned} Error^e &= Error(\boldsymbol{p};\boldsymbol{x}^e) \\ &= \left(\frac{1}{2}\right)\mid \boldsymbol{f}(\boldsymbol{x}^e;\boldsymbol{p})-\boldsymbol{o}^e\mid^2 \end{aligned}$$

We would like to compute

$$\frac{\partial Error^e}{\partial w_{kl,m}}$$

where $w_{kl,m}$ is the m'th component of $\boldsymbol{w}_{kl}$.

Forward pass: Compute $\boldsymbol{f}(\boldsymbol{x}^e;\boldsymbol{p})$, saving all intermediate variables

Backward pass: Compute

$$\begin{aligned} \delta^e_{2l} &= (f_l(\boldsymbol{x}^e)-o^e_l)\phi'_{2l} \\ \phi'_{2l} &= \frac{\partial\phi}{\partial u}\text{ evaluated at }u=\boldsymbol{w}_{21}\cdot\boldsymbol{y} \\ \frac{\partial Error^e}{\partial w_{2l,m}} &= \sum_e\{\delta^e_{2l}(y_m(\boldsymbol{x}^e))\} \end{aligned}$$

Now compute

$$\begin{aligned} \delta^e_{1l} &= \sum_k\{\delta^e_{2k}w_{2k,l}\}\,\phi'_{1l} \\ \phi'_{1l} &= \frac{\partial\phi}{\partial u}\text{ evaluated at }u=\boldsymbol{w}_{11}\cdot\boldsymbol{z} \\ \frac{\partial E^e}{\partial w_{1l,m}} &= \delta^e_{1l}z_m(\boldsymbol{x}^e) \end{aligned}$$

Algorithm 20.8: *Backpropagation to compute the derivative of the fitting error of a two-layer neural net on a single example with respect to its parameters.*

illumination effects look enough like a linear ramp (one side is bright, the other side is dark, and there is a smooth transition between them) that we can simply fit a linear ramp to the intensity values and subtract that from the image window. Another way to do this would be to log-transform the image, and then subtract a linear ramp fitted to the logs. This has the advantage that (using a rather rough model) illumination effects are additive in the log transform. There doesn't appear to be any evidence in the literature that the log transform makes much difference in practice. Another approach is to histogram equalize the window to ensure that its histogram is the same as that of a set of reference images (histogram equalisation is described in section **??**).

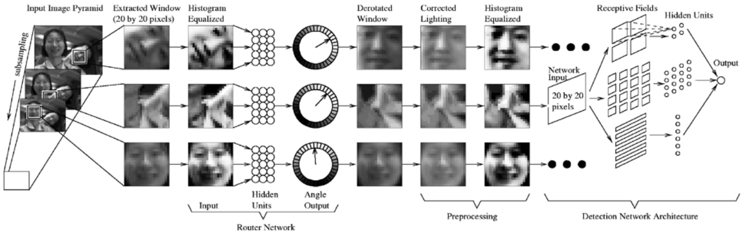

Input Image Pyramid
subsampling
Extracted Window (20 by 20 pixels)
Histogram Equalized
Derotated Window
Corrected Lighting
Histogram Equalized
Receptive Fields
Hidden Units
Output
Network Input
20 by 20 pixels
Input
Hidden Units
Angle Output
Router Network
Preprocessing
Detection Network Architecture

Figure 2. Overview of the algorithm.

Figure 20.13. The architecture of Rowley, Baluja and Kanade's system for finding faces. Image windows of a fixed size are corrected to a standard illumination using histogram equalisation; they are then passed to a neural net that estimates the orientation of the window. The windows are reoriented, and passed to a second net that determines whether a face is present. *figure from Rowley, Baluja and Kanade, Rotation invariant neural-network based face detection p.2, in the fervent hope of receiving permission*

Once the windows have been corrected for illumination, we need to determine whether there is a face present. The orientation isn't known, and so we must either determine it, or produce a classifier that is insensitive to orientation. Rowley, Baluja and Kanade have produced a face finder that finds faces very successfully by firstly estimating the orientation of the window, using one neural net, and then reorienting the window so that it is frontal, and passing the frontal window onto another neural net (see figure 20.13). The orientation finder has 36 output units, each coding for a 10^o range of orientations; the window is reoriented to the orientation given by the largest output. Examples of the output of this system are given in figure 20.14.

20.4.5 Convolutional Neural Nets

Neural networks are not confined to the architecture sketched above; there are a wide variety of alternatives (a good start is to look at []). One architecture that has proven useful in vision applications is the **convolutional neural network**. The motivating idea here is that it appears to be useful to represent image regions with filter outputs. Furthermore, we can obtain a compositional representation we apply

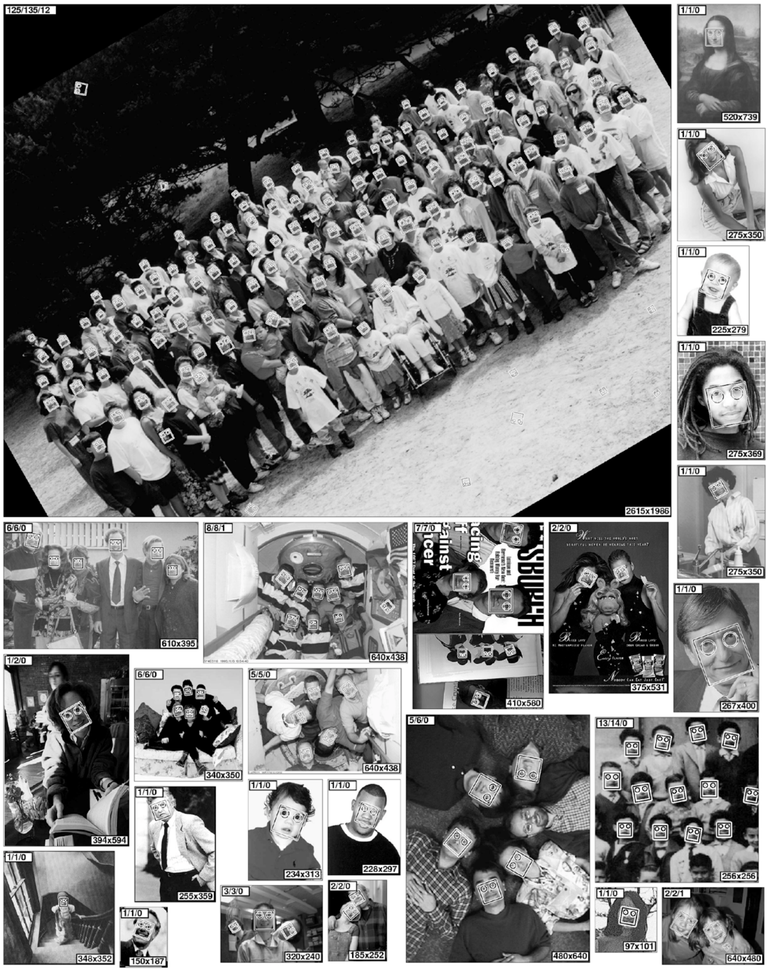

125/135/12
2615x1986
1/1/0
520x739
1/1/0
275x350
1/1/0
225x279
1/1/0
275x369
1/1/0
275x350
6/6/0
610x395
8/8/1
640x438
7/7/0
410x580
2/2/0
375x531
1/1/0
267x400
1/2/0
394x594
6/6/0
340x350
5/5/0
640x438
5/6/0
480x640
13/14/0
256x256
1/1/0
255x359
1/1/0
234x313
1/1/0
228x297
1/1/0
348x352
1/1/0
150x187
3/3/0
320x240
2/2/0
185x252
1/1/0
97x101
2/2/1
640x480

Figure 7. Result of arbitrating between two networks trained with derotated negative examples. The label in the upper left corner of each image (D/T/F) gives the number of faces detected (D), the total number of faces in the image (T), and the number of false detections (F). The label in the lower right corner of each image gives its size in pixels.

Figure 20.14. Typical responses for the Rowley, Baluja and Kanade system for face finding; a mask icon is superimposed on each window that is determined to contain a face. The orientation of the face is indicated by the configuration of the eye-holes in the mask. *figure from Rowley, Baluja and Kanade, Rotation invariant neural-network based face detection p.6, in the fervent hope of receiving permission*

filters to a representation itself obtained using filter outputs. For example, assume that we are looking for handwritten characters; the response of oriented bar filters is likely to be useful here. If we obtain a map of the oriented bars in the image, we can apply another filter to this map, and the output of this filter indicates spatial relations between the bars.

These observations suggest using a system of filters to build up a set of relations between primitives, and then using a conventional neural network to classify on the resulting representation. There is no particular reason to specify the filters in advance; instead, we could learn them too.

Lecun *et al.* have built a number of classifiers for handwritten digits using a convolutional neural network. The basic architecture is given in figure 20.15. The classifier is applied to a 32x32 image window. The first stage — C1 in the figure — consists of six feature maps. The feature maps are obtained by convolving the image with a 5x5 filter kernel, adding a constant, and applying a sigmoid function. Each map uses a different kernel and constant, and these parameters are learned.

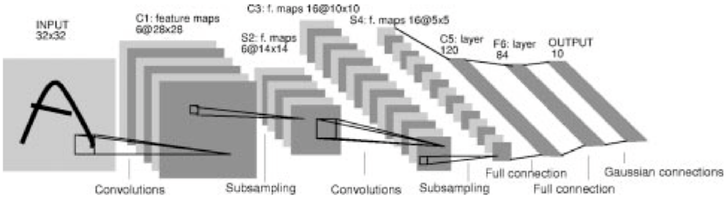


Fig. 2. Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.

Figure 20.15. The architecture of LeNet 5, a convolutional neural net used for recognising handwritten characters. The layers marked C are convolutional layers; those marked S are subsampling layers. The general form of the classifier uses an increasing number of features at increasingly coarse scales to represent the image window; finally, the window is passed to a fully connected neural net, which produces a rectified output that is classified by looking at its distance from a set of canonical templates for characters. *figure from Gradient-Based Learning Applied to Document Recognition Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, p.2283, in the fervent hope of receiving permission*

Because the exact position of a feature should not be important, the resolution of the feature maps is reduced, leading to a new set of six feature maps — S2 in the figure. These maps are, essentially, subsampled versions of the previous layer; this subsampling is achieved by averaging 2x2 neighbourhoods, multiplying by a parameter, adding a parameter, and passing the result through a sigmoid function. The multiplicative and additive parameters are learned. A series of pairs of layers of this form follows, with the number of feature maps increasing as the resolution decreases. Finally, there is a layer with 84 outputs; each of these outputs is supplied by a unit that takes every element of the previous layer as an input.

This network is used to recognise hand printed characters; examples from the training set are given in figure **??**. The outputs are seen as a 7x12 (=84!) image of a character that has been rectified from its hand printed version, and can now be compared with a canonical pattern for that character. The network can rectify distorted characters very successfully (figure **??** shows some extreme cases that are successfully recognised). The input character is given the class of the character whose canonical pattern is closest to the rectified version. The resulting network has a test error rate of 0.95% (figure 20.16).

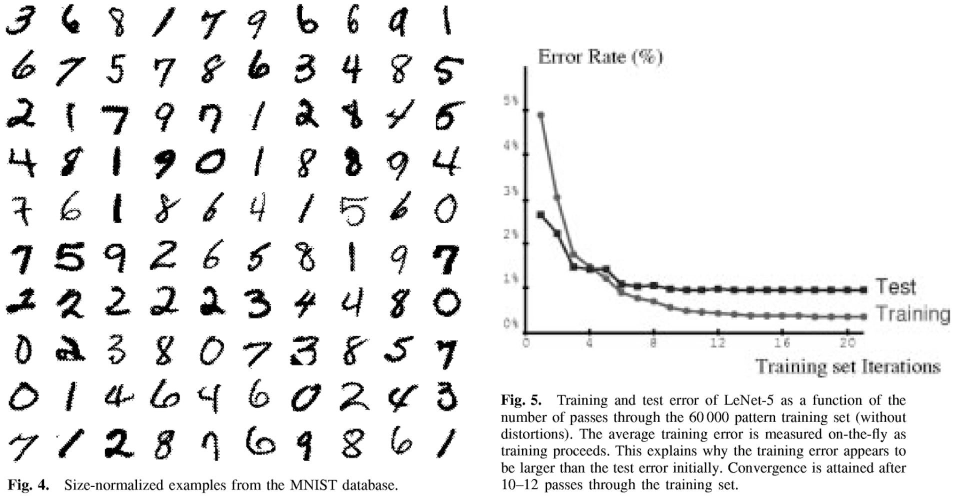


Fig. 4. Size-normalized examples from the MNIST database.

Fig. 5. Training and test error of LeNet-5 as a function of the number of passes through the 60 000 pattern training set (without distortions). The average training error is measured on-the-fly as training proceeds. This explains why the training error appears to be larger than the test error initially. Convergence is attained after 10–12 passes through the training set.

Figure 20.16. On the **left**, a small subset of the MNIST database of handwritten characters, used to train and test LeNet 5. Note the fairly wide variation in the appearance of each character. On the **right**, the error rate of LeNet 5 on a training set and on a test set, plotted as a function of the number of gradient descent passes through the entire training set of 60,000 examples (i.e. if the horizontal axis reads six, the training has taken 360, 000 gradient descent steps). Note that at some point the training error goes down but the test error doesn't; this phenomenon occurs because the system's performance is optimised on the training data. A substantial difference would indicate overfitting. *figure from Gradient-Based Learning Applied to Document Recognition Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, p.2287, in the fervent hope of receiving permission figure from Gradient-Based Learning Applied to Document Recognition Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, p.2288, in the fervent hope of receiving permission*

20.5 The Support Vector Machine

From the perspective of the vision community, classifiers are not an end in themselves, but a means; so when a technique that is simple, reliable and effective becomes available, it tends to be adopted quite widely. The **support vector machine** is such a technique. This should be the first classifier you think of when you wish to build a classifier from examples (unless the examples come from a known distribution, which hardly ever happens). We give a basic introduction to the ideas, and show some examples where the technique has proven useful.

Assume we have a set of N points $\boldsymbol{x}_i$ that belong to two classes, which we shall indicate by 1 and -1. These points come with their class labels, which we shall write as y_i; thus, our data set can be written as

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$$

We should like to determine a rule that predicts the sign of y for any point $\boldsymbol{x}$; this rule is our classifier.

At this point, we distinguish between two cases: either the data is linearly separable, or it isn't. The linearly separable case is much easier, and we deal with it first.

20.5.1 Support Vector Machines for Linearly Separable Datasets

In a linearly separable data set, there is some choice of $\boldsymbol{w}$ and b (which represent a hyperplane) such that

$$y_i\left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) > 0$$

for every example point (notice the devious use of the sign of y_i). There is one of these expressions for each data point, and the set of expressions represents a set of constraints on the choice of $\boldsymbol{w}$ and b. These constraints express the fact that all examples with a negative y_i should be on one side of the hyperplane and all with a positive y_i should be on the other side.

In fact, because the set of examples is finite, there is a family of separating hyperplanes. Each of these hyperplanes must separate the convex hull of one set of examples from the convex hull of the other set of examples. The most conservative choice of hyperplane is the one that is furthest from both hulls. This is obtained by joining the closest points on the two hulls, and constructing a hyperplane perpendicular to this line, and through its midpoint. This hyperplane is as far as possible from each set, in the sense that it maximises the minimum distance from example points to the hyperplane (figure 20.17)

Now we can choose the scale of $\boldsymbol{w}$ and b, because scaling the two together by a positive number doesn't affect the validity of the constraints $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) > 0$. This means that we can choose $\boldsymbol{w}$ and b such that for every data point we have

$$y_i\left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) \geq 1$$

and such that equality is achieved on at least one point on each side of the hyperplane. Now assume that $\boldsymbol{x}_k$ achieves equality and $y_k = 1$, and $\boldsymbol{x}_l$ achieves equality and $y_l = -1$. This means that $\boldsymbol{x}_k$ is on one side of the hyperplane and $\boldsymbol{x}_l$ is on the other; furthermore, the distance from $\boldsymbol{x}_l$ to the hyperplane is minimal (among the points on the same side as $\boldsymbol{x}_l$), as is the distance from $\boldsymbol{x}_k$ to the hyperplane. Notice that there might be several points with these properties.

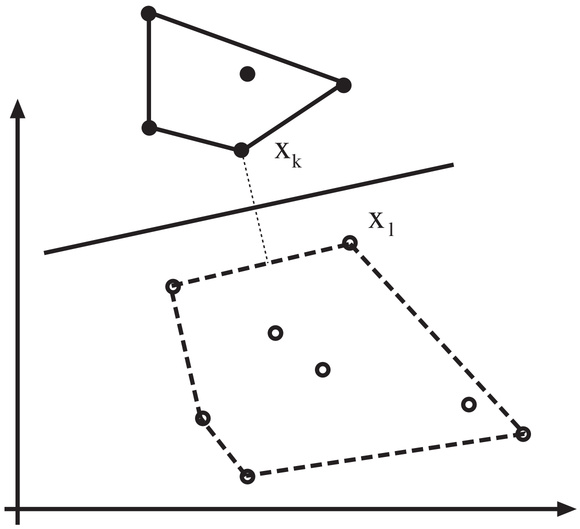


Figure 20.17. The hyperplane constructed by a support vector classifier for a plane data set. The filled circles are data points corresponding to one class, and the empty circles are data points corresponding to the other. We have drawn in the convex hull of each data set. The most conservative choice of hyperplane is one that maximises the minimum distance from each hull to the hyperplane. A hyperplane with this property is obtained by constructing the shortest line segment between the hulls, and then obtaining a hyperplane perpendicular to this line segment and through its midpoint. Only a subset of the data determines the hyperplane. Of particular interest are points on each convex hull which are associated with a minimum distance between the hulls — we will use these points to find the hyperplane in the text.

This means that $\boldsymbol{w} \cdot (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 2$, so that

$$\begin{aligned} dist(\boldsymbol{x}_k, \text{hyperplane}) + dist(\boldsymbol{x}_l, \text{hyperplane}) &= (\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot \boldsymbol{x}_k + \frac{b}{|\boldsymbol{w}|}) - (\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot \boldsymbol{x}_1 + \frac{b}{|\boldsymbol{w}|}) \\ &= \frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot (\boldsymbol{x}_1 - \boldsymbol{x}_2) = \frac{2}{|\boldsymbol{w}|} \end{aligned}$$

This means that maximising the distance is the same as *minimising* $(1/2)\boldsymbol{w} \cdot \boldsymbol{w}$. We now have the constrained minimisation problem:

$$\begin{aligned} \text{minimize} \quad & (1/2)\boldsymbol{w} \cdot \boldsymbol{w} \\ \text{subject to} \quad & y_i\,(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \end{aligned}$$

where there is one constraint for each data point.

Solving for the Support Vector Machine

We can solve this problem, by introducing Lagrange multipliers α_i to obtain the Lagrangian

$$(1/2)\boldsymbol{w}\cdot\boldsymbol{w} - \sum_1^N \alpha_i \left(y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_1 + b\right) - 1\right)$$

This Lagrangian needs to be minimised with respect to $\boldsymbol{w}$ and $\boldsymbol{b}$ and maximised with respect to α_i — these are the Karush-Kuhn-Tucker conditions []. A little manipulation leads to the requirements that

$$\sum_1^N \alpha_i y_i = 0$$

and

$$\boldsymbol{w} = \sum_1^N \alpha_i y_i \boldsymbol{x}_i$$

This second expression is why the device is known as a support vector machine. Generally, it will be the case that the hyperplane is determined by a relatively small number of example points, and the position of other examples is irrelevant (see figure 20.17 — everything inside the convex hull of each set of examples is irrelevant to choosing the hyperplane, and most of the hull vertices are, too). This means that we expect that most α_i are zero, and the data points corresponding to non-zero α_i — which are the ones that determine the hyperplane — are known as the **support vectors**.

Now by substituting these expressions into the original problem and manipulating, we obtain the **dual problem** given by

$$\begin{array}{rl} \text{maximize} & \displaystyle\sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j=1}^N \alpha_i(y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j)\alpha_j \\ \text{subject to} & \alpha_i \geq 0 \\ \text{and} & \displaystyle\sum_{i=1}^N \alpha_i y_i = 0 \end{array}$$

You should notice that the criterion is a quadratic form in the Lagrange multipliers. This problem is a standard numerical problem, known as **quadratic programming**. One can use standard packages quite successfully for this problem, but it does have special features — while there may be a very large number of variables, most will be zero at a solution point — which can be exploited [].

20.5.2 Finding Pedestrians using Support Vector Machines

At a fairly coarse scale, pedestrians have a characteristic, “lollipop-like” appearance — a wide torso on narrower legs. This suggests that they can be found using a

Notation:
We have a training set of N examples

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$$

where y_i is either 1 or -1.

Solving for the SVM:
Set up and solve the dual optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_i^N \alpha_i - \frac{1}{2}\sum_{i,j=1}^N \alpha_i (y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j)\alpha_j \\ \text{subject to} \quad & \alpha_i \geq 0 \\ \text{and} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \end{aligned}$$

We can then determine $\boldsymbol{w}$ from

$$\boldsymbol{w} = \sum_1^N \alpha_i y_i \boldsymbol{x}_i$$

Now for any example point $\boldsymbol{x}_i$ where α_i is non-zero, we have that

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$$

which yields the value of b.

Classifying a point:
Any new data point is classified by

$$\begin{aligned} f(\boldsymbol{x}) &= \operatorname{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b) \\ &= \operatorname{sign}\left(\left(\sum_1^N \alpha_i y_i \boldsymbol{x} \cdot \boldsymbol{x}_i\right) + b\right) \\ &= \operatorname{sign}\left(\sum_1^N (\alpha_i y_i \boldsymbol{x} \cdot \boldsymbol{x}_i + b)\right) \end{aligned}$$

Algorithm 20.9: *Finding an SVM for a Linearly Separable Problem*

support vector machine. The general strategy is the same as for the face-finding example in section **??**: each image window of a fixed size is presented to a classifier, which determines whether the window contains a pedestrian or not. The number of pixels in the window may be large, and we know that many pixels may be irrelevant. In the case of faces, we could deal with this by cropping the image to an oval shape which would contain the face. This is harder to do with pedestrians, because their outline is of a rather variable shape.

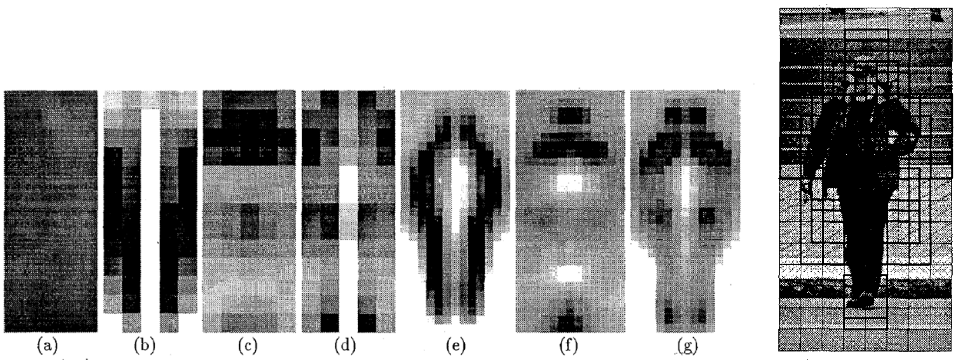


Figure 20.18. On the **left**, averages over the training set of different wavelet coefficients at different positions in the image. Coefficients that are above the (spatial) average value are shown dark, and those that are below are shown light. We expect that noise has the average value, meaning that coefficients that are very light or very dark contain information that could identify pedestrians. On the **right**, a grid showing the support domain for the features computed. Notice that this follows the boundary of the pedestrian fairly closely. *figure from Papageorgiou, Oren and Poggio, A general framework for object detection, p.3, in the fervent hope of receiving permission*

We need to identify features that can help determine whether a window contains a pedestrian or not. It is natural to try to obtain a set of features from a set of examples. A variety of feature selection algorithms might be appropriate here (all of them are variants of search). Papageorgiou, Oren and Poggio chose to look at local features — **wavelet coefficients**, which are the response of specially selected filters with local support — and to use an averaging approach. In particular, they argue that the background in a picture of a pedestrian looks like noise, images that don't contain pedestrians look like noise, and the average noise response of their filters is known. This means that attractive features are ones whose average over many images of pedestrians is different from their noise response. If we average the response of a particular filter in a particular position over a large number of images, and the average is similar to a noise response, then that filter in that position is not particularly informative.

Now that features have been chosen, training follows the lines of section **??**. Papageorgiou, Oren and Poggio use bootstrapping (section **??**), which appears to improve performance significantly.

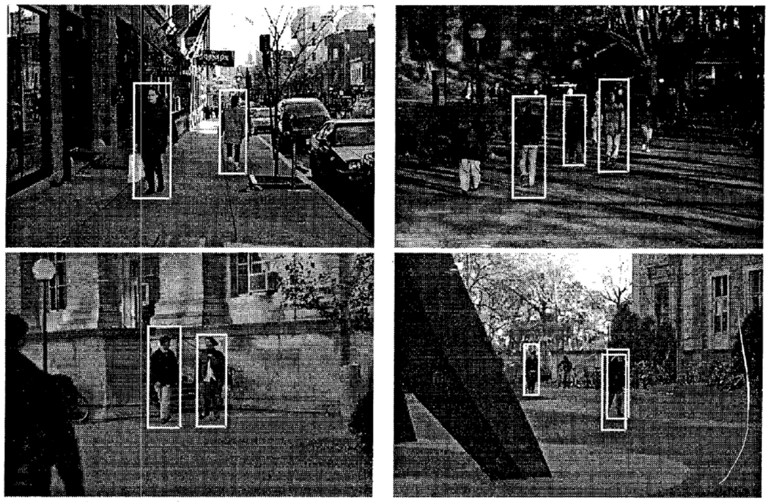
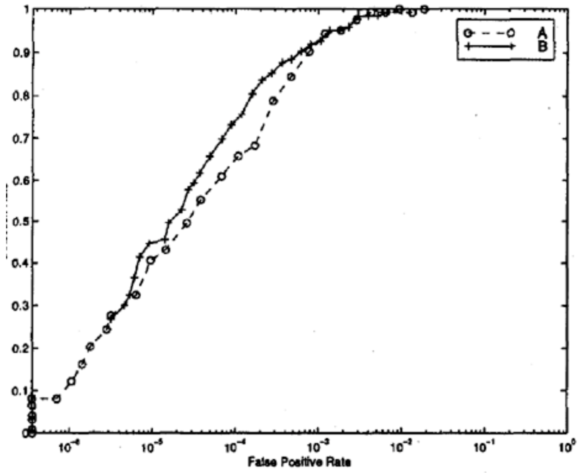


Figure 20.19. Examples of pedestrians detected using the method of Papageorgiou, Oren and Poggio. While not all pedestrians are found, there is a fairly high detection rate. The ROC is in figure 20.20. *figure from Papageorgiou, Oren and Poggio, A general framework for object detection, p.3, in the fervent hope of receiving permission*



(b) People Detection System

Figure 20.20. The receiver operating curve for the pedestrian detection system of Papageorgiou, Oren and Poggio. *figure from Papageorgiou, Oren and Poggio, A general framework for object detection, p.4, in the fervent hope of receiving permission*

20.6 Conclusions

This topic is one on which no orthodoxy is yet established; instead, one tries to use methods that seem likely to work on the problem in hand. For the sake of brevity, we have omitted a vast number of useful techniques; Vapnik's book [?], Bishop's book [?], Ripley's book [?] and McClachlan's book [?] are good places to start.

Choosing a decision boundary is strictly easier than fitting a posterior model. However, with a decision boundary there is no reliable indication of the extent to which an example belongs to one or another class, as there is with a posterior model. Furthermore, fitting a decision boundary requires that we know the classes to which the example objects should be allocated. It is by no means obvious that one can construct an unambiguous class hierarchy for the objects we encounter in recognition problems. Both approaches can require very large numbers of examples to build useful classifiers. Typically, the stronger the model that is applied, the fewer examples required to build a classifier.

It is difficult to build classifiers that are really successful when objects have a large number of degrees of freedom (though see section **??**), and classifiers tend to be difficult to use if the number of features can vary from example to example; in both cases, some form of structural model appears to be necessary. However, estimating, representing and manipulating probability densities in the very high dimensional spaces that occur in vision problems is practically impossible, unless very strong assumptions are applied. Furthermore, it is easy to build probability models for which inference is again practically impossible; it isn't yet known how to build models that are easy to handle of the scale required for vision problems.

This subject is currently at the cutting edge of research in vision and learning. It's hard to know how to choose a method for a given problem, and opportunism seems to be the best approach at present. The examples in this chapter and in the next chapter illustrate a range of approaches that have been taken — some are very successful — but don't yet represent a clear theory.

An alternative approach is to train multiple classifiers and combine their outputs. This strategy is usually known as **boosting**. Boosting is most useful for classifiers with quite simple decision boundaries; these are usually easy to train, but have quite poor performance. Typically, we train a classifier, and then determine the examples in the training set that it gets wrong. These examples are then emphasized — either by weighting errors on them more heavily, or inserting copies into the training set — and a new classifier is trained. We now find out what the new classifier gets wrong, and emphasize these examples and train again; this proces continues through many iterations. Now the outputs of all the classifiers are combined using a set of weights.

Assignments

Exercises

1. Assume that we are dealing with measurements $\boldsymbol{x}$ in some feature space S. There is an open set D where any element is classified as class one, and any element in the interior of $S - D$ is classified as class two.

    - Show that

    $$\begin{aligned} R(s) &= Pr\left\{1 \to 2|\text{using } s\right\} L(1 \to 2) + Pr\left\{2 \to 1|\text{using } s\right\} L(2 \to 1) \\ &= \int_D p(2|\boldsymbol{x}) d\boldsymbol{x} L(1 \to 2) + \int_{S-D} p(1|\boldsymbol{x}) d\boldsymbol{x} L(2 \to 1) \end{aligned}$$

    - Why are we ignoring the boundary of D (which is the same as the boundary of $S - D$) in computing the total risk?

2. In section 2, we said that, if each class-conditional density had the same covariance, the classifier of algorithm 2 boiled down to comparing two expressions that are linear in $\boldsymbol{x}$.

    - Show that this is true.
    - Show that, if there are only two classes, we need only test the sign of a linear expression in $\boldsymbol{x}$.

3. In section 20.3.1, we set up a feature u, where the value of u on the i'th data point is given by $u_i = \boldsymbol{v} \cdot (\boldsymbol{x}_i - \boldsymbol{\mu})$. Show that u has zero mean.

4. In section 20.3.1, we set up a series of features u, where the value of u on the i'th data point is given by $u_i = \boldsymbol{v} \cdot (\boldsymbol{x}_i - \boldsymbol{\mu})$. We then said that the $\boldsymbol{v}$ would be eigenvectors of Σ, the covariance matrix of the data items. Show that the different features are independent, using the fact that the eigenvectors of a symmetric matrix are orthogonal.

5. In section **??**, we said that the ROC was invariant to choice of prior. Prove this.

Programming Assignments

1.

II Appendix: Support Vector Machines for Datasets that are not Linearly Separable

In many cases, a separating hyperplane will not exist. To allow for this case, we introduce a set of **slack variables**, $\xi_i \geq 0$, which represent the amount by which the constraint is violated. We can now write our new constraints as

$$y_i\left(\boldsymbol{w} \cdot \boldsymbol{x}_1 + b\right) \geq 1 - \xi_i$$

and we modify the objective function to take account of the extent of the constraint violations, to get the problem

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} & y_i\left(\boldsymbol{w} \cdot \boldsymbol{x}_1 + b\right) \geq 1 - \xi_i \\ \text{and} & \xi_i \geq 0 \end{array}$$

Here C gives the significance of the constraint violations with respect to the distance between the points and the hyperplane. The dual problem becomes

$$\begin{array}{rl} \text{maximize} & \sum_{i}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i (y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j) \alpha_j \\ \text{subject to} & C \geq \alpha_i \geq 0 \\ \text{and} & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{array}$$

Again, we have

$$\boldsymbol{w} = \sum_{1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

but recovering b from the solution to the dual problem is slightly more interesting. For each example where $C > \alpha_i > 0$ (note that these are strict inequalities, unlike the constraints) the slack variable ξ_i will be zero. This means that

$$\sum_{j=1}^{N} y_j \alpha_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j + b = y_i$$

for these values of i. This expression yields b. Again, the optimization problem is a quadratic programming problem, though there is no guarantee that many points will have $\alpha_i = 0$.

III Appendix: Using Support Vector Machines with Non-Linear Kernels

For many data sets, it is unlikely that a hyperplane will yield a good classifier. Instead, we want a decision boundary with a more complex geometry. One way to achieve this is to map the feature vector into some new space, and look for a hyperplane in that new space. For example, if we had a plane data set that we were convinced could be separated by plane conics, we might apply the map

$$(x, y) \to (x^2, xy, y^2, x, y)$$

to the dataset. A classifier boundary that is a hyperplane in this new feature space is a conic in the original feature space. In this form, this idea is not particularly useful, because we might need to map the data into a very high dimensional space (for example, assume that we know the classifier boundary has degree two, and the data is 10 dimensional — we would need to map the data into a 65 dimensional space).

Write the map as $\boldsymbol{x}' = \phi(\boldsymbol{x})$. Write out the optimisation problem for the new points $\boldsymbol{x}'_i$; you will notice that the only form in which $\boldsymbol{x}'_i$ appears is in the terms

$$\boldsymbol{x}'_i \cdot \boldsymbol{x}'_j$$

which we could write as $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$. Apart from always being positive, this term doesn't give us much information about ϕ. In particular, the map doesn't appear explicitly in the optimisation problem. If we did solve the optimisation problem, the final classifier would be

$$\begin{aligned} f(\boldsymbol{x}) &= \operatorname{sign}\left(\sum_1^N (\alpha_i y_i \boldsymbol{x}' \cdot \boldsymbol{x}'_i + b)\right) \\ &= \operatorname{sign}\left(\sum_1^N (\alpha_i y_i \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}_i) + b)\right) \end{aligned}$$

Assume that we have a function $k(\boldsymbol{x}, \boldsymbol{y})$ which is positive for all pairs of $\boldsymbol{x}$, $\boldsymbol{y}$. It can be shown that, under various technical conditions of no interest to us, there is some ϕ such that $k(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. All this allows us to adopt a clever trick — instead of constructing ϕ explicitly, we obtain some appropriate $k(\boldsymbol{x}, \boldsymbol{y})$, and use it in place of ϕ. In particular, the dual optimisation problem becomes

$$\begin{aligned} \text{maximize} \quad & \sum_i^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i (y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)) \alpha_j \\ \text{subject to} \quad & \alpha_i \geq 0 \\ \text{and} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \end{aligned}$$

and the classifier becomes

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{1}^{N}(\alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b)\right)$$

Of course, these equations assume that the dataset are separable in the new feature space represented by k. This may not be the case, in which case the problem becomes

$$\begin{aligned} \text{maximize} \quad & \sum_{i}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i (y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j))\alpha_j \\ \text{subject to} \quad & C \geq \alpha_i \geq 0 \\ \text{and} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

and the classifier becomes

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{1}^{N}(\alpha_i y_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b)\right)$$

There are a variety of possible choices for $k(\boldsymbol{x}, \boldsymbol{y})$. The main issue is that it must be positive for all values of $\boldsymbol{x}$ and $\boldsymbol{y}$. Some typical choices are shown in table 20.1. There doesn't appear to be any principled method for choosing between kernels; one tries different forms, and uses the one that gives the best error rate, measured using cross-validation.

Kernel form	Qualitative properties of ϕ represented by this kernel
$(\boldsymbol{x} \cdot \boldsymbol{y})^d$	ϕ is all monomials of degree d
$(\boldsymbol{x} \cdot \boldsymbol{y} + c)^d$	ϕ is all monomials of degree d or below
$\tanh(a\boldsymbol{x} \cdot \boldsymbol{y} + b)$	
$\exp(-\frac{(\boldsymbol{x}-\boldsymbol{y})^T(\boldsymbol{x}-\boldsymbol{y})}{2\sigma^2})$	

Table 20.1. Some support vector kernels

Chapter 21

RECOGNITION BY RELATIONS BETWEEN TEMPLATES

An object may have internal degrees of freedom, which mean that its appearance is highly variable — for example, people can move arms and legs; fish deform to swim; snakes wriggle; etc. This phenomenon can make template matching extremely difficult, because one may require either a classifier with a very flexible boundary (and a lot of examples) or many different templates.

One useful strategy is to find small components of the object — which will have a fairly orderly appearance — and match these components as templates. We could then determine what objects are present by looking for suggestive relationships between the templates that have been found. For example, instead of finding a face by looking for a single complete face template, we could find one by looking for eyes, nose and a mouth that all lie in an appropriate configuration.

This approach has several possible advantages. Firstly, it may be easier to learn an eye template than it is to learn a face template, because the structure could be simpler. Secondly, it may be possible to obtain and use relatively simple probability models. This is because there may be some independence properties that can be exploited. Thirdly, we may be able to match a very large number of objects with a relatively small number of templates. Animal faces are a good example of this phenomenon — pretty much all animals with recognisable faces have eyes, nose and a a mouth, but with slightly different spatial layouts. Finally, it means that the simple individual templates can be used to construct complex objects. For example, people can move their arms and legs around, and it appears to be much more difficult to learn a single, explicit template for finding whole people than to obtain individual templates for bits of people and a probability model that describes their degrees of freedom.

21.1 Finding Objects by Voting on Relations between Templates

Very simple object models can result in quite effective recognition. The simplest model is to think of an object as a collection of image **patches** — small image neighbourhoods of characteristic appearance — of several different types, forming an image **pattern**. To tell what pattern is present in an image, we find each patch that is present, and allow it to vote for every pattern in which it appears. The pattern in the image is the one with the most votes. While this strategy is simple, it is quite effective. We will sketch methods for finding patches, and then describe a series of increasingly sophisticated versions of the strategy.

21.1.1 Describing Image Patches

Small image patches can have a quite characteristic appearance, usually when they have many non-zero derivatives (e.g. at corners). We describe a system due to Schmid and Mohr that takes advantage of this property []. They find image corners — often called **interest points** — (using a corner detector like that of section **??**, in []); they then estimate a set of derivatives of the image grey-level at those corners, and evaluate a set of functions of the image derivatives that are invariant to rotation, translation, some scaling, and illumination changes. These features are called **invariant local jets**. Describing these features in detail would take us out of our way. The value of this approach is that, because the combinations are invariant, we expect that they will take the same value on different views of an object.

We now assume that the image patches fall into a number of classes. We can obtain representatives for each class by having multiple pictures of each object — typically, corresponding patches will be of the same class, but probably have somewhat different invariant local jets, as a result of image noise. We can determine an appropriate set of classes either by classifying the patches by hand, or by clustering example patches (a somewhat better method!). We need to be able to tell when two sets of invariant local jets represent the same class of image patch. Schmid and Mohr test the Mahalanobis distance between the feature vectors of a patch to be tested and an example patch; if it is below some threshold, the patch being tested is the same as the example.

Notice that this is a classifier — it is allocating patches to classes represented by the examples, or deciding not to classify them — and that the patches are templates. We could build a template matcher with any of the techniques of that chapter without using features that are invariant to rotation, etc. To do this, we would use a training set that contained rotated and scaled versions of each example patch, under varying illuminant conditions, so the classifier could learn that rotation, scaling and illuminant changes don't affect the identity of the patch. The advantage of using invariant features is that the classifier doesn't need to learn this invariance from the training set.

21.1.2 Voting and a Simple Generative Model

For a given image, we find the interest points and classify the image patch at each image point. Now, what pattern lies in the image? we can answer this question by constructing a correspondence between image patches and patterns. Assume that there are N_i patches in the image. Furthermore, we assume that there is either a single pattern from our collection in the image, or there is no pattern there. An individual patch could have come either from whatever pattern is present, or from noise. However, patterns typically do not contain every class of patch. This means that asserting that a particular pattern is present is the same as asserting that some of the image patches came from noise (because only one pattern can be present, and these images patches belong to classes that are not in the current pattern).

We have a (very simple) generative model for an image. When a pattern is present, it produces patches of some classes, but not others. By elaborating this model, we obtain a series of algorithms for matching patterns to images.

The simplest version of this model is obtained by assuming that a pattern produces all patches of the classes that it can produce, and then require that as few patches as possible come from noise. This assumption boils down to voting. We take each image patch, and record a vote for every pattern that is capable of producing that class of patch. The pattern with the most votes, wins, and we say that this pattern is present. This strategy can be effective, but has some problems (figure 21.3).

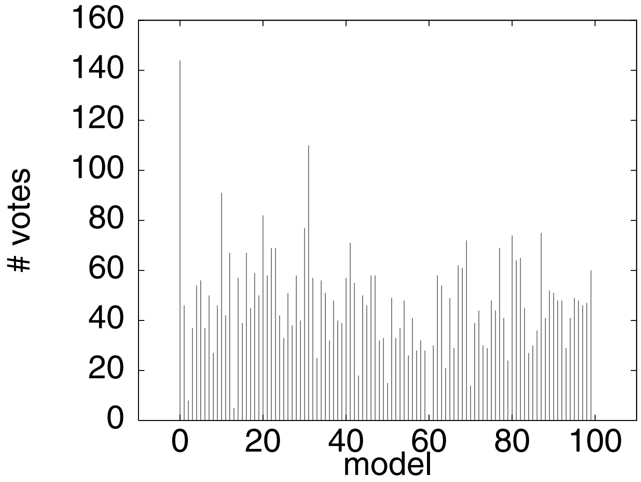


Figure 21.1. The graph shows the number of votes for each pattern recorded for a particular image under the simple voting scheme. Notice that, while the correct match (model # 0) receives the maximum number of votes, three other candidates receives more than half as many votes. Reprinted from [?], Figure 3.

21.1.3 Probabilistic Models for Voting

We can interpret our simple voting process in terms of a probabilistic model. This is worth doing, because it will cast some light on the strengths and weaknesses of the approach. Our generative model can be made probabilistic by assuming that the patches are produced independently and at random, assuming that the object is present. Let us write

$$P\{\text{patch of type } i \text{ appears in image}|j\text{'th pattern is present}\} = p_{ij}$$

and

$$P\{\text{patch of type } i|\text{no pattern is present}\} = p_{ix}$$

In the very simplest model, we assume that, for each pattern j, we assume that $p_{ij} = \mu$ if the pattern can produce this patch and 0 otherwise. Furthermore, we assume that $p_{ix} = \lambda < \mu$ for all i. Finally, we assume that each observed patch in the image can come from either a single pattern or from noise. There are a total of n_i patches in the image. Under these assumptions, we need only know which patches came from a pattern and which from noise to compute a likelihood value. In particular, The likelihood of the image, given a particular pattern, and assuming that n_p patches came from that pattern and $n_i - n_p$ patches come from noise, is

$$P(\text{interpretation}|\text{pattern}) = \lambda^{n_p}\mu^{(n_i-n_p)}$$

and this value is larger for larger values of n_p. However, because not every pattern can produce every image patch, the maximum available choice of n_p is *dependent on the pattern we choose.* Our voting method is equivalent to choosing the pattern with the maximum possible likelihood under this (very simple) generative model.

This suggests the source of some difficulties: if the pattern is unlikely, we should take that (prior) information into account. Furthermore, noise may be able to produce some patches more easily than others — ignoring this fact can confuse the vote. Finally, some patches may be more likely given an object than others. For example, corners appear much more often on a checkerboard pattern than they do on a zebra stripe pattern.

Elaborating the Generative Model

Dealing with these issues in the framework of our current model — that the patches occur independently given that the pattern is present — is relatively simple. Assume that there are N different types of patch. We now assume that each different type of patch is generated with a different probability by different patterns, and by noise. Now assume that there are n_{ij} instances of the j'th type of patch in the image. Furthermore, n_k of these are generated by the pattern, and the rest are generated by noise.

The likelihood function for the l'th pattern is

$$P\left(\begin{array}{c|c} n_1 \text{ of type 1 from pattern,} & \\ \ldots, & \\ n_N \text{ patches of type } N \text{ from pattern} & j\text{'th pattern} \\ \text{and } n_{i1}-n_1 \text{ of type 1 from noise,} & \\ \ldots, & \\ n_{iN}-n_N \text{ from noise} & \end{array}\right)$$

Now because the patches arise independently given the pattern and the noise is independent of the pattern, this likelihood is

$$P(\text{patches from pattern}|j\text{'th pattern})P(\text{patches from noise})$$

The first term is

$$P(\text{type } 1|j\text{'th pattern})^{n_1}P(\text{type } 2|j\text{'th pattern})^{n_2}\ldots P(\text{type } N|j\text{'th pattern})^{n_N}P(\text{noise})$$

which is evaluated as

$$p_{1j}^{n_1}p_{2j}^{n_2}\ldots p_{Nj}^{n_N}$$

We now assume that patches that arise from noise do so independently of one another. This means we can write the noise term as

$$P(\text{type } 1|\text{noise})^{(n_{i1}-n_1)}\ldots P(\text{type } N|\text{noise})^{(n_{iN}-n_N)}$$

which is evaluated as

$$p_{1x}^{(n_{i1}-n_1)}\ldots p_{Nx}^{(n_{iN}-n_N)}$$

This means that the likelihood can be written out as

$$p_{1j}^{n_1}p_{2j}^{n_2}\ldots p_{Nj}^{n_N}p_{1x}^{(n_{i1}-n_1)}\ldots p_{Nx}^{(n_{iN}-n_N)}$$

There are two cases for each type of patch k; if $p_{kj} > p_{kx}$, then this is maximised by $n_k = n_{ik}$, otherwise it is maximised by $n_k = 0$. We write π_j for the prior probability that the image contains the j'th pattern and π_0 for the prior probability it contains no object. This means that for each object type j, the maximal value of the posterior will look like

$$\left(\prod_m p_{mj}^{n_{im}} \prod_l p_{lx}^{n_{il}}\right)\pi_j$$

where m runs through features for which $p_{mj} > p_{mx}$ and l runs through features for which $p_{lj} < p_{lx}$. We could form this value for each object, and choose the one with the highest posterior value. Notice that this *is* a relational model, though we're not actually computing geometric features linking the templates. This is because the patches are linked by the conditional probability that they occur, given a pattern, *which is different from pattern to pattern.* You should compare this model with the face detector of section **??**; the probabilistic models are identical in spirit.

21.1.4 Voting on Relations

We can use geometric relations to improve on the simple voting strategy fairly easily. A patch should match to an object only if it there are nearby patches that also match to the object, and are in an appropriate configuration. The term "appropriate configuration" can be a source of difficulties, but for the moment assume that we are matching objects up to *plane* rotation, translation and scale (this is a reasonable assumption for such things as frontal faces).

Now assume that we have a patch that matches to some object. We now take the p nearest patches, and check firstly that more than 50% of them match to the same objects, and secondly that the angles between the triples of matching patches are the same as the corresponding angles on the object — these are referred to as **semilocal constraints** (see figure 21.2). If these two tests are passed, we register a vote for that object from the patch. You should compare this strategy — which is due to Schmid and Mohr [] — quite closely with geometric hashing, section **??**. It is rather hard to construct a probabilistic interpretation for this approach.

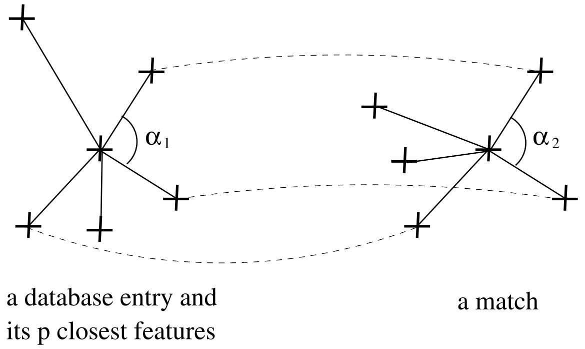


Figure 21.2. Instead of voting for patches that match, we can vote for collections of patches. In this approach, we register a vote only if a patch matches to a particular object, and a percentage of its neighbours match too, and the angles between triples of matching patches have appropriate values, as the figure illustrates. This strategy significantly reduces the number of random matches, as figure 21.3 indicates. Reprinted from [**?**], Figure 4.

21.1.5 Voting and 3D Objects

Although we described Schmid and Mohr's approach in terms of 2D patterns, it can be extended fairly easily to 3D object recognition. We do this by regarding each of a series of views of the object as a different 2D pattern. Given enough views, this will work, because the small changes in the 2D pattern caused by a slight change in viewing angle will be compensated for by the available error range in the process that matches invariant local jets and angles.

This strategy for turning 3D recognition into 2D matching applies quite gen-

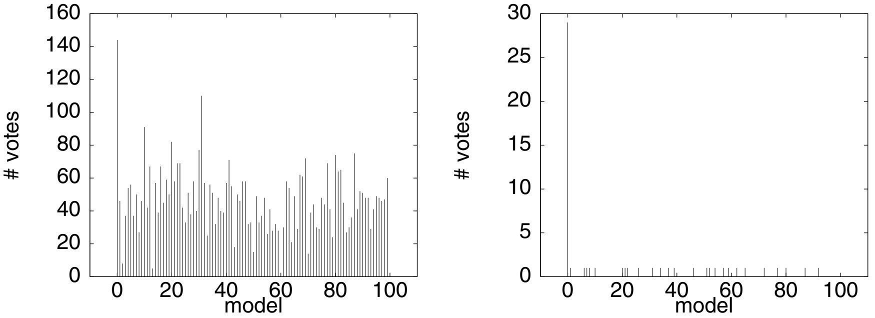


Figure 21.3. In practice, the use of semilocal constraints greatly increases the discriminating power of the original voting scheme. The figure on the **left** shows the number of votes recorded for each model under the original voting scheme, for a particular image. Although the correct match (model # 0) receives the maximum number of votes, three other candidates receives more than half as many votes. When semilocal constraints are added (**right**), the correct match stands out much more clearly. Reprinted from [**?**], Figures 3 and 5.

erally, but comes with difficulties. The main problem is the very large number of models that result, which can make the voting procedure difficult. It isn't known what the minimum number of views required for matching in a scheme like this is. Figure 21.4 illustrates a matching result obtained using this approach.

21.2 Relational Reasoning using Probabilistic Models and Search

The previous section explored methods that assumed that templates were conditionally independent given the pattern. This assumption is a nonsense for most objects (though it can work extremely well in practice) because there are usually quite strong relations between features. For example, there are very seldom more than two eyes, one nose and one mouth in a face; the distance between the eyes is roughly the same as that from the bridge of the nose to the mouth; the line joining the eyes is roughly perpendicular to the line from bridge of nose to mouth; etc. We incorporated some these constraints into a voting strategy in section 21.1.4, but didn't really indicate any principled framework within which they can be exploited.

This remains a difficult problem. We need to build models that represent what is significant and allow for efficient inference. We can't present current orthodoxy on this subject, because it doesn't exist; instead, we will describe the main issues and approaches.

21.2.1 Correspondence and Search

In this section, we will explore the core issues in using a probabilistic model for matching. The general approach is as follows. We obtain an interpretation for the image, and compute the value of the posterior probability of the interpretation

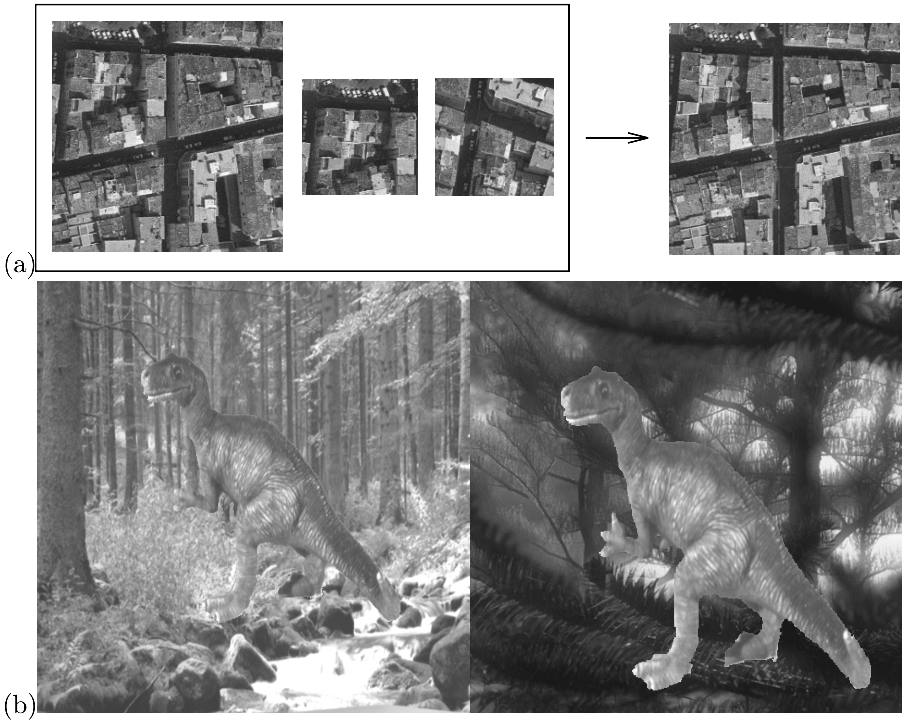


Figure 21.4. Recognition results: (a) image matching in aerial photo interpretation: the image on the right is correctly retrieved using any of the images on the left; (b) three-dimensional object recognition: a toy dinosaur is correctly recognized in both images, despite a large amount of background clutter. Reprinted from [**?**], Figures 8 and 9.

given the image. We accept interpretations with a sufficiently large value of the posterior probability.

This account hides a myriad of practical problems. The most basic is that we do not know which image information comes from objects and which comes from noise. Generally, we will evade this difficulty and turn our problem into a correspondence problem by matching templates to the image, and then reasoning about the relations between templates. For example, if we wish to find faces, we will apply a series of different detectors — say eye, nose and mouth detectors — to the image, and then look for configurations that are suggestive.

Correspondence

This brings us to the second important problem, which is correspondence. We can't evaluate a posterior (or joint) probability density if we don't know the values of each variable. This means that we must, in essence, engage in a process that hypothesizes that one response is the left eye, another the right eye, a third the nose and a fourth the mouth, and *then* evaluates the posterior. It is clearly important to manage this search very carefully indeed, so that we do not have to look at all possible correspondences.

The techniques of chapter **??** transfer relatively easily; we need to apply a cloak of probability, but the basic line of reasoning will remain. The most basic fact of object recognition — for rigid objects, a small number of correspondences can be used to generate a large number of correspondences — translates easily into the language of probability easily, where it reads: for rigid objects, correspondences are not independent.

In general, our probability model will evaluate the joint probability density for some set of variables. We call a set of values for these variables an **assembly** (other terms are a **group**, or an **hypothesis**). An assembly consists of a set of detector outputs — each may respond with position, or position and orientation, or even more — and a label for each output. These labels give the correspondences, and the labelling is important: if, say, the eye-detector does not differentiate between a left eye and a right eye, we will have to label the eye responses left and right, but there is no point in labelling a mouth detector response as an eye.

It is not possible to form and test all assemblies, because there are usually far too many. For example, assume we have eye detectors — which don't differentiate between left and right eyes — nose detectors and mouth detectors, and a face consists of two eyes, a nose and a mouth. If there are N_e responses from eye detectors, N_n responses from nose detectors and N_m responses from mouth detectors, we would have $O(N_e^2 N_n N_m)$ assemblies to look at. If you read chapter **??** carefully, you should be convinced this is a wild overestimate — the whole point of the chapter is that quite small numbers of correspondences predict other correspondences.

Incremental Assembly and Search

This suggests thinking of the matching process in the following way. We will assemble collections of detector responses, labelling them as necessary to establish their role in the final assembly. The assembly process will be incremental — we will expand small assemblies to make big ones. We will take each working assembly, and then determine whether: it can be pruned; it can be accepted *as is*; or it should be expanded. Any particular assembly consists of a set of pairs, each of which is a detector response and a label. Some available labels may not have a detector response associated with them. Generally, a correspondence search will have a working collection of hypotheses, which may be quite large. The search involves: taking some correspondence hypothesis, attaching some new pairs, and then either accepting the result as an object, removing it from consideration entirely, or returning it to the pool of working hypotheses.

There are three important components in a correspondence search:

- **When to accept an hypothesis:** if a correspondence hypothesis is sufficiently good, it may be possible to stop expanding it.

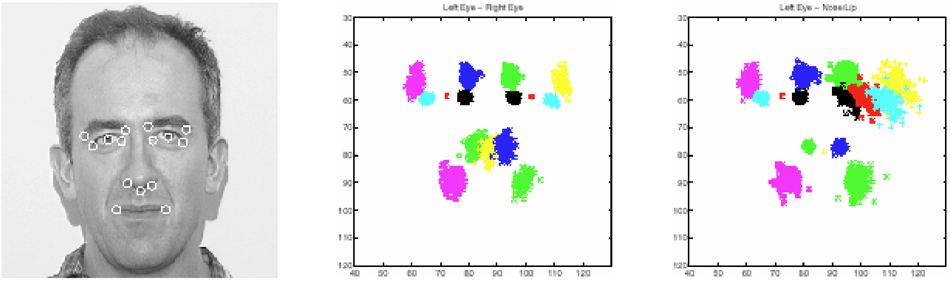


Figure 21.5. Perona's feature detector responses

- **What to do next:** in a correspondence search, we have to determine which correspondence hypothesis to work on next. Generally, we would wish to work on an hypothesis that is likely to succeed. Usually, it is easier to determine what *not* to do next.

- **When to prune an hypothesis:** if there is no set of detector responses that can be associated with any subset of the empty labels such that the resulting hypothesis would pass the classifier criterion, then there is no point in expanding that search.

21.2.2 Example: Finding Faces

Perona and colleagues have built a series of face finders that use various forms probability model. Each follows broadly the line of incremental search amongst assemblies, identifying assemblies that should be expanded. None prunes the search. Typical outputs for their feature detectors are shown in figure 21.22. They compute invariants by using reference points (which set up a canonical coordinate system, allowing invariants to be computed). As figure 21.22 indicates, the choice of reference points significantly affects the variance of the coordinates computed with respect to those reference points. This is because the relative configuration of the eyes is more stable than that of, say, the left eye and the lip.

Figure 21.23 indicates how the detectors respond to a typical face image. All these responses are searched; there is no pruning, and instead of classifying hypotheses, assemblies are ranked. The best matching assembly is a fair guide to the presence and position of a face (figure 21.24).

Setting up the Problem

We will continue to work with the face model, to illustrate the ideas. Assume that the left eye detector responds at $\boldsymbol{x}_1$, the right eye detector responds at $\boldsymbol{x}_2$, the mouth detector responds at $\boldsymbol{x}_3$ and the nose detector responds at $\boldsymbol{x}_4$; we assume that we believe the face is at $\boldsymbol{F}$, and that all other detector responses are due to noise. Furthermore, we assume that we are determining whether there is either a

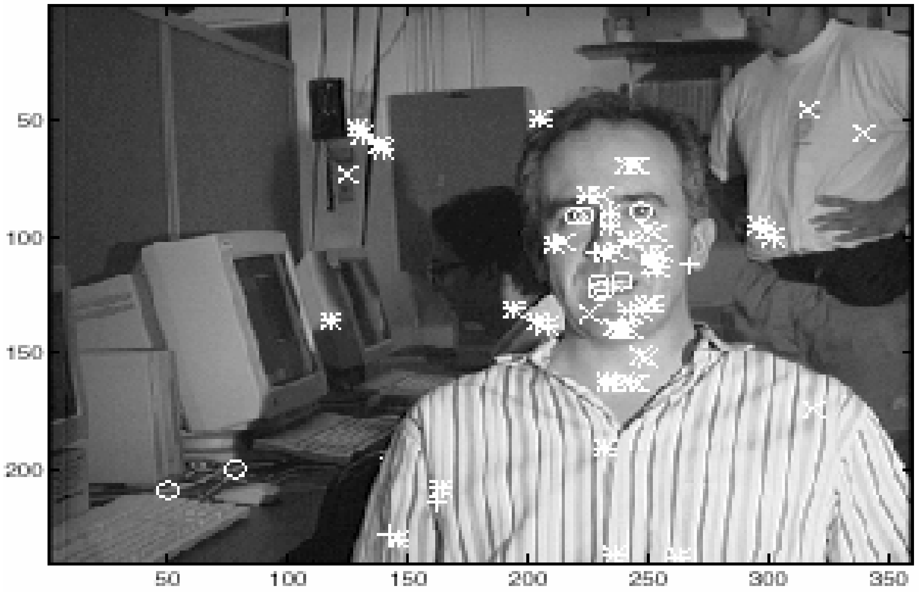

50
100
150
200
50
100
150
200
250
300
350

Figure 21.6. Perona's feature detector outputs

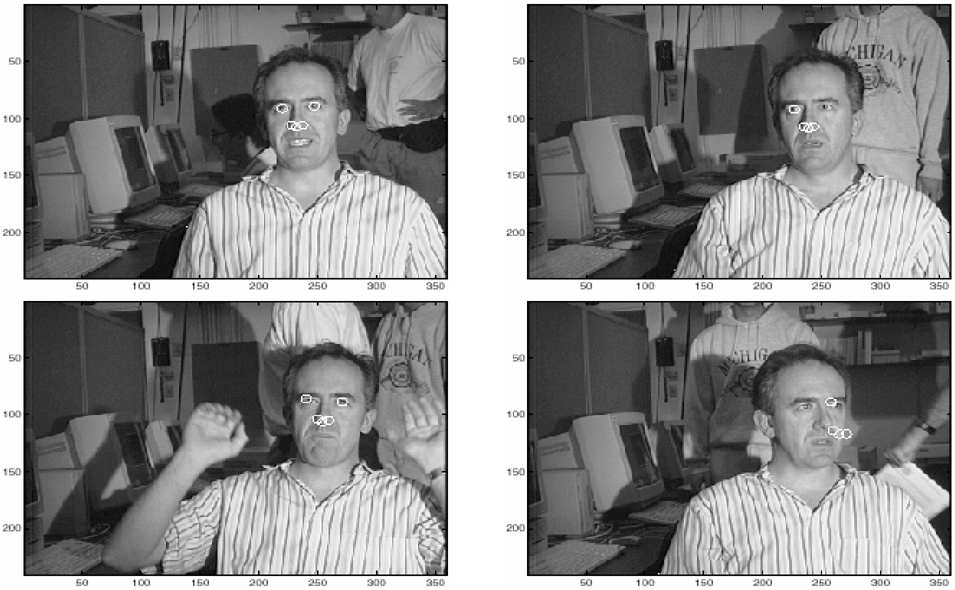


Figure 21.7. Perona's matcher

single face or none in the image (or window!). This involves comparing the value of

$$P(\text{one face at } \boldsymbol{F}|\boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses})$$

with

$$P(\text{no face}|\boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses})$$

Stopping a Search: Detection

Assume that noise responses are independent of the presence of a face (which is fairly plausible). We can then write

$$\begin{aligned} P(\text{one face at } \boldsymbol{F}|\boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses}) = \\ P(\text{one face at } \boldsymbol{F}|\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4)P(\text{all other responses}) \propto \\ P(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})P(\text{all other responses})P(\text{one face at } \boldsymbol{F}) \end{aligned}$$

(where we have suppressed some notation!). We can classify particular groups of detector responses as coming from a face or from noise by comparing the posterior that this configuration comes from a face with the posterior that it comes from noise. In particular, we will compare

$$P(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})$$

with

$$(P(\text{noise responses})P(\text{no face})/P(\text{one face at } \boldsymbol{F}))\,(\text{term in relative loss})$$

(recall that there are only two options — face or no face — and check chapter **??** again if the remark seems puzzling). Thus, the likelihood is the main term of interest. This tells us whether a complete assembly represents a face; but an incomplete assembly could represent a face, too. We can score configurations that lack features, too. This involves determining the posterior that a face is present given only some features, and comparing that with the posterior that the features arose from noise.

When a group of features satisfies the classification criterion (that the posterior that a face is present exceeds the posterior that it is not), we can certainly stop searching. It may not be necessary to observe all possible features to determine that a face is present. If a configuration is strongly suggestive of a face and unlikely to have arisen from noise, then we may wish to assert that a face is present and stop searching at that point.

We illustrate with an example; assume we wish to determine whether a right eye, a mouth and a nose represent a face. To evaluate the joint, we will need to evaluate a series of noise terms and the term

$$P(\boldsymbol{X}_{\text{le}} = \text{missing}, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})$$

using our model. This requires a model to explain how a feature went missing; the simplest is to assume that the detector did not respond (a variety of others are possible), and that this failure to respond is independent of the other feature detector responses. This yields

$$P(\text{missing}, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) = \int P(\text{le does not respond}|\boldsymbol{X}_1) P(\boldsymbol{X}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) d\boldsymbol{X}_1$$

(again, suppressing some notation). Now if the probability of detector failure is independent of feature position (which is the usual case in vision applications), we have

$$P(\text{missing}, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) = P(\text{le does not respond}) P(\boldsymbol{X}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) d\boldsymbol{X}_1$$

This *does not mean* that we can prune a search if a configuration doesn't represent a face when it has a missing feature. This is because there might be some position for the missing feature that did represent a face; as section **??** will show, to prune we need a bound. If we do decide that an object is present without matching all components, this is because either (a) missing components didn't get detected, but the configuration of the components found is so distinctive it doesn't matter or (b) we haven't yet found the other components, but the configuration of the other detectors is so distinctive it doesn't matter. Notice that in case (b), we may still want to look for possible responses for the other detectors; exercise **??** discusses how deciding that an object is present changes this search process.

21.3 Using Classifiers to Prune Search

Assume that we have an assembly that consists of a right eye, a mouth and a nose. If we can determine from the probability model that there is no possible position for the left eye that would result in an assembly that is acceptable as a face, then there is no point in trying to grow our assembly. Recall that, if we had a position for the left eye, we would test to see whether the likelihood was greater than some function of the number of noise responses, the priors, and the relative losses. In particular, there is some fixed value that the likelihood must exceed before we can assert that a face is present. If we can assert that *there is no value of* $\boldsymbol{x}_1$ that would result in a likelihood that is over threshold, the search can be pruned.

Notice that this line of reasoning extends. If we have a left eye response and a right eye response, and there is no value of the nose and mouth responses that would result in a posterior that is over threshold, we can prune this search — and so not attempt to add either a nose or a mouth to this assembly. It is often quite tricky to supply the necessary bounds, however. One way to do this is to use a classifier.

You can think of a classifier as representing a (very crude) probability model for the posterior. This model has a zero value when the posterior is below threshold, and a non-zero value when it is above. The great advantage of this model is that it is quite easy to prune — we can reject any assembly if there was no element that could be added to yield something that would lie in the non-zero region.

21.3.1 Identifying Acceptable Assemblies Using Projected Classifiers

There is no point in growing a working assembly unless there is some conceivable way in which it could become an acceptable final assembly. Assume that we have a classifier that can determine whether a final assembly is acceptable or not. This classifier can be trained on a large sequence of examples.

We can use this final classifier to predict tests that determine whether small groups should be expanded or not — essentially, we discard groups for which there is no prospect that any conceivable new set of elements would make the group acceptable. In turn, this boils down to projecting the decision boundaries of the classifier onto a set of different factors. For example, assume that we have a two eye detector responses — should they form a pair or not? We can look at only features that come from those eyes, because we don't know what other elements lie in the assembly, and therefore can't use them to compute features. In particular, the issue here is: is there some set of elements (i.e. a nose and a mouth) such that, if they were attached to this group, the resulting group would pass the classifier? This question is answered by a classifier whose decision boundary is obtained by projecting the decision boundary of the face assembly onto the space spanned by the features computed from our two eyes (as in figure 21.8).

By projecting the classifier onto components that can handle small groups, we can prune the collection of groups that must be searched to find a large collection of segments that passes the main classifier. Using this strategy in practice requires a certain amount of care. It is important to have classifiers which project well — the decision boundaries must project to decision boundaries that can be represented reasonably easily. This can be dealt with — the details are rather beyond the scope of this account — and the resulting classifiers can be used to find people and horses relatively efficiently in quite simple cases (figure 21.9 and figure 21.11).

21.3.2 Example: Finding People and Horses Using Spatial Relations

Assume we wish to find people in images; a natural approach is to find possible body segments, and reason about their configuration. This can be done in some specialised cases. One that may become important is if the people concerned are not wearing clothing (which would allow people to search for or to avoid images that are associated with strong opinions). In this case, body segments can be found by looking for skin (as in section **??**), and then constructing extended image regions with roughly parallel sides (as in section **??**) that contain skin colour. The resulting extended image regions tend to represent many of the body segments in the image.

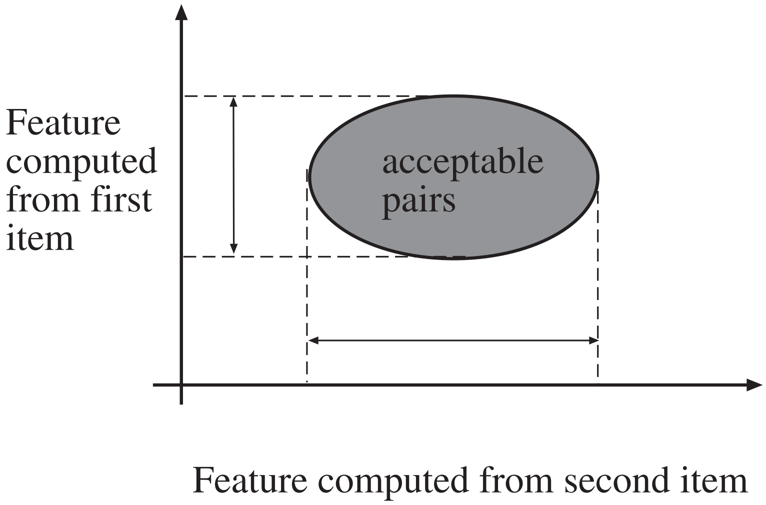

Feature computed from first item
acceptable pairs
Feature computed from second item

Figure 21.8. A classifier that accepts large groups can be used to predict tests that identify potentially useful small groups. Assume we want to group two items together: this grouping is acceptable only if a point, consisting of one feature computed from each item lies, in the shaded set. There are some items that could not appear in these groups, because the feature computed from the items lies out of the acceptable range, so that there does not exist a second item that could make the pair acceptable. It is possible to obtain classifiers that identify the acceptable range by projecting the main classifier onto its factors, as the picture indicates.

Now we can search for people by searching for assemblies of these image segments. For example, assume that we will see only frontal views; then we can look for assemblies of nine image segments, containing a left upper arm, a left lower arm, etc. and a torso. It isn't practical to take all sets of nine segments from the image, so we use the methodology above. This yields a system that can find people in simple images fairly reliably (figure **??**).

This methodology extends to finding horses as well. We identify all image pixels that could be hide (this means they lie in the right range of colours, and have little texture locally). We then form extended regions of hide (as for people), and regard these as potential body segments. We use a crude model of a horse as a four segment group (a body, a neck and two legs); the order in which the tests are performed is illustrated in figure 21.10. The crude model is used because (a) it is robust to changes in aspect and (b) it doesn't require accurate localisation of all segments.

This method of finding objects has been useful in some applications, as figure 21.9 and figure 21.11 suggest, but there are significant open questions. Firstly, it isn't obvious what sequence of intermediate groups and tests is best. It is possible to think about this problem by asking which projections of the decision boundary lead to small projected volumes (and so, hopefully, to relatively few passing groups).

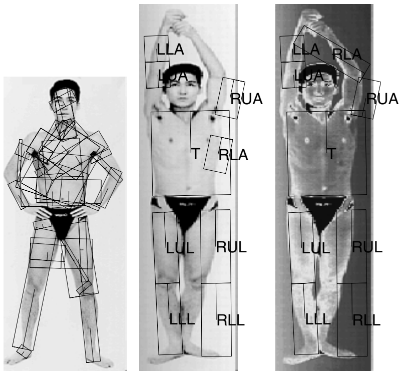


Figure 21.9. People can be found by looking for assemblies of extended image segments — those that look like cylinders — that, taken together, look like a person. The system illustrated here looks for assemblies of nine segments (left and right upper and lower arms and legs, and a torso). It isn't possible to look at all possible nine segment groups, because there are too many, so the system expands only groups that could pass the final classifier if appropriate segments were attached. This test is implemented using the mechanism described in the text. The figure on the **left** shows a set of extended segments extracted for an image containing a person. At the **center** and on the **right**, labeled segment configurations that could correspond to a person, where `T`=torso, `LUA`=left upper arm, etc. The head is not marked because we are not looking for it with our method. A single leg segment has been broken to generate the upper and lower leg segments. Notice that, while the assemblies that don't look like people have been pruned, that doesn't guarantee that a person leads to a single assembly.

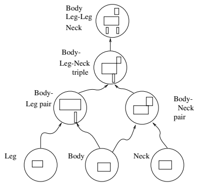


Figure 21.10. The model used for a horse involves two legs, a body and a neck. Segments are first tested to see whether they are acceptable members of one of these classes. We then test pairs of body and leg segments, and pairs of body and leg segments. We then take body-leg pairs and body-neck pairs *which share the same body* and test to see if they form a valid body-leg-neck triple. Pairs of triples that *share the same body and the same neck* are tested to form quadruples.

This looks like a difficult problem, because it deals with combinatorial properties. Secondly, it isn't obvious how we deal with multiple objects efficiently. If a single template occurs only on one object, then, when we observe the template, we immediately have an object hypothesis. Things are much more difficult when templates could have come from many objects. For example, consider recognizing people *and* horses: a single segment could come from either, as could many pairs — how do we arrange the sequence of tests to minimize the work required to decide what we are dealing with? Finally, the use of a classifier generates some annoying problems with false negatives, which we discuss further in section **??**.

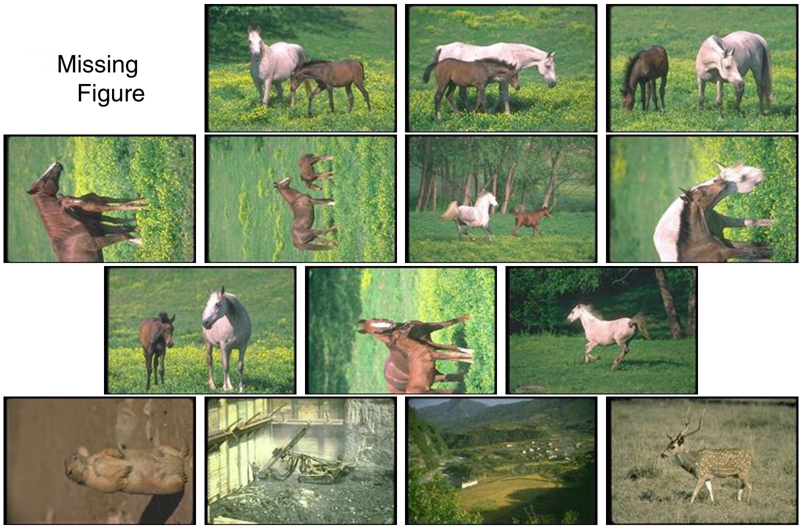


Figure 21.11. Horses can be found by building a classifier of the type described in the text. This example illustrates the effectiveness of the approach. The system finds image regions that have the colour and texture of hide, and look like cylinders (as in figure **??**, section **??**). A horse is defined as a collection of four cylinders — a body, a neck and two legs — and a classifier that accepts this assembly is learned from examples. This classifier is projected onto factors, as described in the text, and these projected versions are used to build assemblies in an order that was arbitrarily selected. The result is a system that, while not spectacularly accurate, is useful. The figure shows all images that the classifier thinks contains horses, obtained using a control set of 1086 non-horse images and a test set of 100 horse images. The test images recovered contain horses in a wide range of aspects; one control image contains an animal that might reasonably pass for a horse.

21.4 Technique: Hidden Markov Models

Up to this point, adopting a probability model hasn't changed much of significance in our discussion of recognition. While we perform "verification" by evaluating a joint probability model, we are still engaged in a correspondence search, very like those of chapter **??**. Our method of pruning is analogous to the interepretation tree of section **??**. However, more is possible: some probability models have a structure that make it possible to get an exactly optimal correspondence very efficiently indeed.

A program that reads American Sign Language from a video sequence of someone signing must infer a state, internal to the user, for each sign. The program will infer state from measurements of hand position that are unlikely to be accurate, but will depend — hopefully quite strongly — on the state. The signs change state in a random (but quite orderly) fashion. In particular, some sequences of states occur very seldom (e.g. a sequence of letter signs for the sequence “wkwk” is extremely unlikely). This means that both the measurements *and* the relative probabilities of different sequences of signs can be used to determine what actually happened.

The elements of this kind of problem are:

- there is a sequence of random variables (in our example, the signs), each of which is conditionally independent of all others given its predecessor;
- each random variable generates a measurement (the measurements of hand position) whose probability distribution depends on the state.

Similar elements are to be found in such examples as interpreting the movement of dancers, or of martial artists. There is an extremely useful formal model, known as a **hidden Markov model**, corresponding to these elements.

The sequence of random variables does not have to be temporal. Instead, we could order the variables by spatial relations, too. Consider an arm: the configuration of the lower arm is (roughly!) independent of the configuration of the rest of the body given the configuration of the upper arm; the configuration of the upper arm is (roughly!) independent of the rest of the body given the configuration of the torso; etc. This gives us a sequence of random variables with the conditional independence properties described above. Now we don’t really know the configuration of the lower arm — we have only some image measurements that have some probabilistic relationship with the configuration of the lower arm (as above).

The sequence of random variables is often thought of as a state model, where at each tick of a clock, the state changes randomly; this is helpful for the sign language example, but not as helpful for the arm model.

21.4.1 Formal Matters

A sequence of random variables X_n is said to be a **Markov chain** if

$$P(\boldsymbol{X}_n = \boldsymbol{a}|\boldsymbol{X}_{n-1} = \boldsymbol{b}, \boldsymbol{X}_{n-2} = \boldsymbol{c}, \ldots, \boldsymbol{X}_0 = \boldsymbol{x}) = P(\boldsymbol{X}_n = \boldsymbol{a}|\boldsymbol{X}_{n-1} = \boldsymbol{b})$$

and a **homogenous Markov chain** if this probability does not depend on n. Markov chains can be thought of as sequences with very little memory; the new state depends on the previous state, but not on the whole history. It turns out that this property is surprisingly useful in modelling, because many physical variables appear to have it, and because it enables a variety of simple inference algorithms. There are very slightly different notations for Markov chains on discrete and continuous state spaces.

Hidden Markov Models on Discrete State Spaces

Assume that we have a discrete state space. It doesn't really matter what dimension the space is, although finite spaces are somewhat easier to imagine. Write the elements of the space as s_i and assume that there are k elements. Assume that we have a sequence of random variables taking values in that state space that forms a homogenous Markov chain. Now we write

$$P(X_n = s_j | X_{n-1} = s_i) = p_{ij}$$

and because the chain is independent of n, so is p_{ij}. We can write a matrix $\mathcal{P}$ with i, j'th element p_{ij} which describes the behaviour of the chain; this matrix is called the **state transition matrix**. Assume that X_0 has probability distribution $P(X_0 = s_i) = \pi_i$, and we will write $\boldsymbol{\pi}$ as a vector with i'th element π_i. This means that

$$\begin{aligned} P(X_1 = s_j) &= \sum_{i=1}^{k} P(X_1 = s_j | X_0 = s_i) P(X_0 = s_i) \\ &= \sum_{i=1}^{k} P(X_1 = s_j | X_0 = s_i) \pi_i \\ &= \sum_{i=1}^{k} p_{ij} \pi_i \end{aligned}$$

and so the probability distribution for the state of X_1 is given by $\mathcal{P}^T \boldsymbol{\pi}$. By a similar argument, the probability distribution for the state of X_n is given by $(\mathcal{P}^T)^n \boldsymbol{\pi}$. For all Markov chains, there is at least one distribution $\boldsymbol{\pi}^s$ such that $\boldsymbol{\pi}^s = \mathcal{P}^T \boldsymbol{\pi}^s$; this is known as the **stationary distribution** of the chain. Markov chains allow quite simple and informative pictures. We can draw a weighted, directed graph with a node for each state and the weight on each edge indicating the probability of a state transition (figure 21.12).

If we observe the random variable X_n, then inference is easy — we know what state the chain is in. This is a poor observation model, however. A much better model is to say that, for each element of the sequence, we observe *another* random variable, whose probability distribution depends on the state of the chain. That is, we observe some Y_n, where the probability distribution is some $P(Y_n | X_n = s_i) = q_i(Y_n)$. We can arrange these elements into a matrix $\mathcal{Q}$. Specifying a hidden Markov model requires providing the state transition process, the relationship between state and the probability distribution on Y_n and the initial distribution on states, i.e. the model is given by $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})$. We will assume that the state space has k elements.

21.4.2 Computing with Hidden Markov Models

We will assume that we are dealing with a hidden Markov model on a discrete state space — this simplifies computation considerably, usually at no particular cost. There are two important problems:

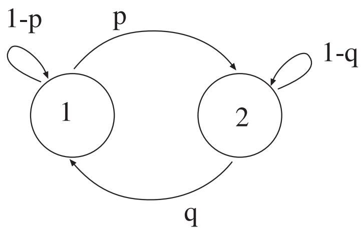


Figure 21.12. A simple, two-state Markov chain. In this chain, the probability of going from state one to state two is p; from state one to state one is $1 - p$; etc. We could describe this chain with the state transition matrix Its stationary distribution is $(q/(p+q), p/(p+q))$. This makes sense; for example, if p is very small and q is close to one, the chain will spend nearlly all its time in state one. Notice that, if p and q are both very small, the chain will stay in one state for a long time, and then flip to the other state, where it will stay for a long time.

- **Inference:** We need to determine what underlying set of states gave rise to our observations. This will make it possible to, for example, infer what the dancer is doing or the signer is saying.
- **Fitting:** We need to choose a hidden Markov model that represents a sequence of past observations well.

Each has an efficient, standard solution.

The Trellis Model

Assume that we have a series of N measurements $\boldsymbol{Y}_i$ that we believe to be the output of a hidden Markov model. We can set up these measurements in a structure called a **trellis**. This is a weighted, directed graph consisting of N copies of the state space, which we arrange in columns. There is a column corresponding to each measurement. We weight the node representing state $\boldsymbol{X}_i$ in the column corresponding to $\boldsymbol{Y}_j$ with $\log q_i(\boldsymbol{Y}_j)$.

We join up the elements from column to column as follows. Consider the column corresponding the $\boldsymbol{Y}_j$; we join the element in this column representing state $\boldsymbol{X}_k$ to the element in the column corresponding to $\boldsymbol{Y}_{j+1}$ representing state $\vec{X}_l$ if p_{kl} is non-zero. This arc represents the fact that there is a possible transition between these states. This arc is weighted with $\log p_{kl}$. Figure 21.13 shows a trellis constructed from an HMM.

The trellis has the following interesting property: each (directed) path through the trellis represents a legal sequence of states. Now since each node of the trellis is weighted with the log of the emission probability, and each arc is weighted with the log of the transition probability, the likelihood of sequence of states can be obtained by identifying the path corresponding to this sequence, and summing the weights (of arcs and nodes) along the path. This yields an extremely effective algorithm for

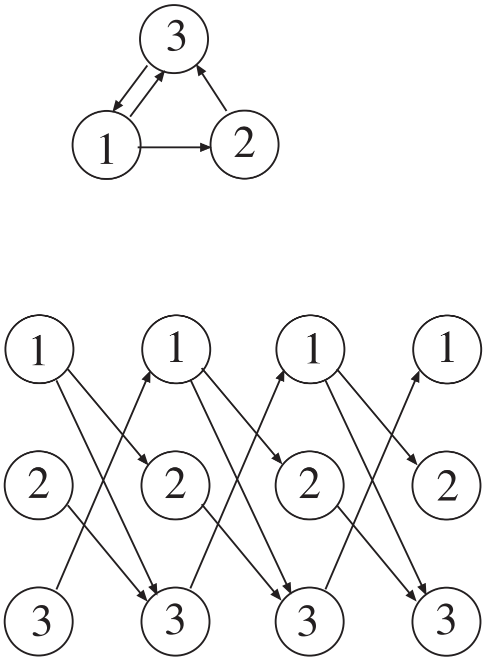


Figure 21.13. *At the top, a simple state transition model. We have labelled each arc with the log of the probability that that transition occurs from the relevant start node. Below, the trellis corresponding to that model. Notice that each path through the trellis corresponds to a legal sequence of states, for a sequence of four measurements. We weight the arcs with the log of the transition probabilities, and the nodes with the log of the emission probabilities; arc weights are not shown here, to reduce the complexity of the drawing.*

finding the maximum likelihood path, known as **dynamic programming** or **the Viterbi algorithm**.

We start at the *final* column of the tellis. We know the log-likelihood of a one state path, ending at each node, as this is just the weight of that node. Now consider a two state path, which will start at the second last column of the trellis. We can easily obtain the best path leaving each node in this column. Consider a node: we know the weight of each arc leaving the node and the weight of the node at the far end of the arc, so we can choose the path segment with the largest value of the sum — this arc is the best we can do leaving that node. Now for each node, we add the weight at the node to the value of the best path segment leaving that node (i.e. the arc weight plus the weight of the node at the far end). This sum is the best value obtainable on reaching that node — which we'll call the **node value**.

Now, since we know the best value obtainable on reaching each node in the second-last column, we can figure out the best value obtainable on reaching each node in the third-last column. At each node in the third last column, we have a choice of arcs, each reaching a node whose value we know. We choose the arc with the largest value of (arc weight plus node value), add this value to the weight at the starting node in the third last column, and this yields the value of the starting node. We can repeat this process, until we have a value for each of the nodes in the first column; the largest value is the maximum likelihood.

We can also get the path with the maximum likelihood value. When we compute the value of a node, we erase all but the best arc leaving that node. Once we reach the first column, we simply follow the path from the node with the best value. Figure 21.14 illustrates this extremely simple and very powerful algorithm.

In the following sections, we describe dynamic programming rather more formally.

Inference and Dynamic Programming

For inference, we have a series of observations $\{Y_0, Y_1, \ldots, Y_n\}$ and we would like to obtain the sequence of $n+1$ states $\boldsymbol{S} = \{S_0, S_1, \ldots, S_n\}$ that maximises

$$P(\boldsymbol{S} | \{Y_0, Y_1, \ldots, Y_n\}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

which is the same as maximising the joint distribution

$$P(\boldsymbol{S}, \{Y_0, Y_1, \ldots, Y_n\} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

There is a standard algorithm for this purpose, the **Viterbi algorithm**. We seek an $n+1$ element path through the states (from S_0 to S_n). There are k^{n+1} such paths, because we could choose from each state for every element of the path (assuming that there are no zeros in $\mathcal{P}$ — in most cases, there are certainly $O(k^{n+1})$ paths). We can't look at every path, but in fact we don't have to. The approach is as follows: assume that, for each possible state s_l, we know the value of the joint for the best n step path that ends in $S_{n-1} = s_l$; then the path that maximises the joint for an $n+1$ step path must consist of one of these paths, combined with another step. All we have to do is find the missing step.

We can approach finding the path with the maximum value of the joint as an induction problem. Assume that, for each value j of S_{n-1}, we know the value of the joint for the best path that ends in $S_{n-1} = j$, which we write as

$$\delta_{n-1}(j) = \max_{S_0, S_1, \ldots, S_{n-2}} P(\{S_0, S_1, \ldots, S_{n-1} = j\}, \{Y_0, Y_1, \ldots, Y_{n-1}\} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

Now we have that

$$\delta_n(j) = \left(\max_i \delta_{n-1}(i) P_{ij} \right) q_j(Y_n)$$

We need not only the maximum *value*, but also the path that gave rise to this value. We define another variable

$$\psi_n(j) = \arg\max \left(\delta_{n-1}(i) P_{ij} \right)$$

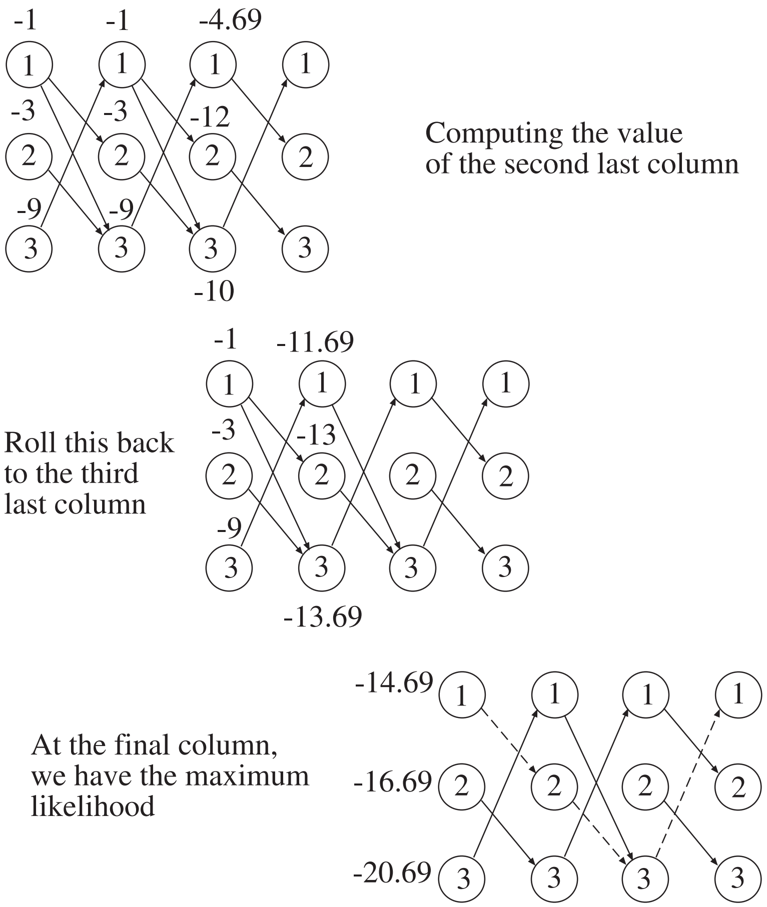

-1
-1
-4.69
1
1
1
1
-3
-3
-12
2
2
2
2
-9
-9
3
3
3
3
-10
Computing the value
of the second last column
-1
-11.69
1
1
1
1
Roll this back
to the third
last column
-3
-13
2
2
2
2
-9
3
3
3
3
-13.69
-14.69
1
1
1
1
At the final column,
we have the maximum
likelihood
-16.69
2
2
2
2
-20.69
3
3
3
3

Figure 21.14. *It is a simple matter to find the best path through the trellis of figure 21.13 (or any other trellis, for that matter!). We compute the value of each node in the second-last column; then of each node in the third last column; etc. as described in the text. Once we get to the start of the trellis, the largest weight is the maximum of the log-likelihood; since we erased all but the best path segments, we have the best path, too (indicated by a dashed line).*

(i.e. the best path that ends in $S_n = j$). This gives us an inductive algorithm for getting the best path.

The reasoning is as follows: I know the best path to each state for the $n-1$'th measurement; for each state for the n'th measurement, I can look backward and choose the best state for the $n-1$'th measurement; but I know the best path from there, so I have the best path to each state for the n'th measurements. We have put everything together in algorithm 1.

1. **Initialization:**

$$\begin{aligned} \delta_1(j) &= \pi_j b_j(Y_1) 1.0\mathrm{cm} 1 \leq j \leq N \\ \psi_1(j) &= 0 \end{aligned}$$

2. **Recursion:**

$$\begin{aligned} \delta_n(j) &= \left(\max_i \delta_{n-1}(i) P_{ij} \right) q_j(Y_n) \\ \psi_n(j) &= \arg\max \left(\delta_{n-1}(i) P_{ij} \right) \end{aligned}$$

3. **Termination:**

$$\begin{aligned} p^* &= \max_i \left(\delta_N(i) \right) \\ q_N^* &= \arg\max_i \left(\delta_N(i) \right) \end{aligned}$$

4. **Path backtracking:**

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$

Algorithm 21.1: The Viterbi algorithm *yields the path through an HMM that maximises the joint, and the value of the joint at this path. Here* δ *and* ψ *are convenient bookkeeping variables (as in the text);*p^* *is the maximum value of the joint; and* q_t^* *is the t'th state in the optimal path.*

Fitting an HMM with EM

We have a dataset $\boldsymbol{Y}$ for which we believe a hidden Markov model is an appropriate model; but which hidden Markov model should we use? We wish to choose a model that best represents a set of data. To do this, we will use a version of the Expectation-Maximisation algorithm of section **??**, due to Baum and Welch []. In this algorithm, we assume that we have an HMM, $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$; we now want to use this model and our dataset to estimate a new set of values for these parameters. We now estimate $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}}))$ using a procedure that is given below. There will be two possibilities (a fact which we won't prove). Either $P(\boldsymbol{Y}|(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}})) > P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})$, or $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}}) = (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})$.

The updated values of the model parameters will have the form:

$$\begin{aligned} \overline{\pi_i} &= \text{expected frequency of being in state } s_i \text{ at time 1} \\ \overline{p_{ij}} &= \frac{\text{expected number of transitions from } s_i \text{ to } s_j}{\text{expected number of transitions from state } s_i} \\ \overline{q_j(k)} &= \frac{\text{expected number of times in } s_j \text{ and observing } Y = y_k}{\text{expected number of times in state } s_j} \end{aligned}$$

We need to be able to evaluate these expressions. In particular, we need to be able to determine

$$P(X_t = s_i, X_{t+1} = s_j|\boldsymbol{Y}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

which we shall write as $\xi_t(i,j)$. If we know $\xi_t(i,j)$, we have

$$\begin{aligned}
\text{expected number of transitions from } s_i \text{ to } s_j &= \sum_{t=1}^{T} \xi_t(i,j) \\
\text{expected number of times in } s_i &= \text{expected number of transitions from } s_i \\
&= \sum_{t=1}^{T}\sum_{j=1}^{N} \xi_t(i,j) \\
\text{expected frequency of being in } s_i \text{ at time 1} &= \sum_{j=1}^{N} \xi_1(i,j) \\
\text{expected number of times in } s_i \text{ and observing } Y = y_k &= \sum_{t=1}^{T}\sum_{j=1}^{N} \xi_t(i,j)\delta(Y_t, y_k)
\end{aligned}$$

where $\delta(u,v)$ is one if its arguments are equal and zero otherwise.

To evaluate $\xi_t(i,j)$, we define a **backward variable** $\beta_t(j) = P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\}|X_t = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$. This backward variable can also be obtained by induction as:

$$\begin{aligned}
\beta_N(j) &= P(\text{no further output}|X_n = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\beta_t(j) &= P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\}|X_t = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \sum_{l=1}^{k} [P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\}, X_t = s_l|X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))] \\
&= \left[\sum_{l=1}^{k} P(X_t = s_l, Y_{t+1}|X_{t+1} = s_j)\right] P(\{Y_{t+2}, \ldots, Y_n\}|X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \left[\sum_{l=1}^{k} p_{jl} q_l(Y_{t+1})\right] \beta_{t+1}(j) \qquad 1 \le t \le k-1
\end{aligned}$$

In terms of the backward variable, we have that

$$\begin{aligned}
\xi_t(i,j) &= P(X_t = s_i, X_{t+1} = s_j|\boldsymbol{Y}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \frac{P(\boldsymbol{Y}, X_t = s_i, X_{t+1} = s_j|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))}{P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\left\{\begin{array}{c} P(Y_0, Y_1, \ldots, Y_t, X_t = s_i|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))\times \\ P(Y_{t+1}|X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))\times \\ P(X_{t+1} = s_j|X_t = s_i, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))\times \\ P(Y_{t+2}, \ldots, Y_N|X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \end{array}\right\}}{P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))} \\
&= \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))} \\
&= \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}
\end{aligned}
$$

As a result, we have a simple fitting algorithm, collected in algorithm 2

```
Until $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_{i+1}$ is the same as $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_i$
  compute the forward variables $\alpha$ and $\beta$
  using the procedures of algorithms 4 and 5

  compute $\xi_t(i,j) = \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}$

  compute the updated parameters using the procedures of algorithm 3

  These values are the elements of $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_{i+1}$
end
```

Algorithm 21.2: Fitting Hidden Markov Models *to a data sequence* $\boldsymbol{Y}$ *is achieved by a version of EM. We assume a model* $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_i$*, and then compute the coefficients of a new model; this iteration is guaranteed to converge to a local maximum of* $P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$.

21.4.3 Varieties of HMM's

We have not spoken about the topology of the graph underlying our model. This could be a **complete graph** (one where every node is connected to every other in both directions), but does not have to be. One disadvantage of using complete graphs is that there are a large number of parameters to estimate. In applications, a number of variants have proven useful.

A **left-right model** or **Bakis model** has the property that, from state i, the model can move only to states $j > i$. This means that for $j < i$, $p_{ij} = 0$. Furthermore, $\pi_1 = 1$ and for all $i \neq 1$, $p_i = 0$. This means that the state transition

$$
\begin{aligned}
\overline{\pi_i} &= \texttt{expected frequency of being in state } s_i \texttt{ at time 1} \\
&= \sum_{j=1}^{N} \xi_1(i,j) \\
\overline{p_{ij}} &= \frac{\texttt{expected number of transitions from } s_i \texttt{ to } s_j}{\texttt{expected number of transitions from state } s_i} \\
&= \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)} \\
\overline{q_i(k)} &= \frac{\texttt{expected number of times in } s_i \texttt{ and observing } Y = y_k}{\texttt{expected number of times in state } s_i} \\
&= \frac{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)\delta(Y_t, y_k)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)}
\end{aligned}
$$

`here` $\delta(u,v)$ `is one if its arguments are equal and zero otherwise.`

Algorithm 21.3: Computing the new values of parameters *for the hidden Markov model fitting process*

$$
\begin{aligned}
\alpha_0(j) &= \pi_j q_j(Y_0) \\
\alpha_{t+1}(j) &= \left[\sum_{l=1}^{k} \alpha_t(l) p_{lj} \right] q_j(Y_{t+1}) \qquad 1 \leq t \leq n-1
\end{aligned}
$$

Algorithm 21.4: Computing the forward variable *for the hidden Markov model fitting process.*

matrix $\mathcal{P}$ will be upper triangular. Furthermore, an N state left-right model will stay in the N'th state once it has been reached. You should notice that this model captures a notion of the order in which events can occur, which could be convenient if we were using HMM's to encode various kinds of motion information (or speech information, where the model originates).

However, a general left-right model suggests that large numbers of events may be skipped (the state may advance freely). This isn't really consistent with a reasonable model of motion, because while we may miss a measurement or two or a state or two, it is unlikely that we will miss a large collection of states. It is common to use

$$
\begin{aligned}
\beta_N(j) &= 1 \\
\beta_t(j) &= \left[\sum_{l=1}^{k} p_{jl} q_l(Y_{t+1})\right] \beta_{t+1}(j) \qquad 1 \leq t \leq k-1
\end{aligned}
$$

Algorithm 21.5: Computing the backward variable *for the hidden Markov model fitting process*

the additional constraint that for $j > i + \delta$, $p_{ij} = 0$. Here δ tends to be a small number (two is often used).

Surprisingly, constraining the topology of the model does not create any problems with the algorithms of the previous sections. You should verify that the estimation algorithm preserves zeros, meaning that if we start it with model of a particular topology — which will have zeros in particular spots of the state transition matrix — it will produce a new estimate of the state transition matrix that also has zeros in those spots.

21.5 Application: Hidden Markov Models and Sign Language Understanding

Sign language is language which is rendered using a system of gestures, rather than by manipulating the vocal tract. For most people, learning to understand sign language takes an effort; it would be attractive to have a device that could be pointed at someone who was signing, and would generate speech. This is a problem that has some strong analogies with speech recognition, an area that is now quite highly developed.

Hidden Markov models have been highly successful in speech understanding applications. The hidden states describe the vocal system; the observations are various acoustic measurements. Typically, there is a hidden Markov model associated with each word. These models are attached together using a **language model**, which specifies the probability of a word occurring given some other word has already occurred. The resulting object is another — possibly very big — HMM. The sentence represented by a set of acoustic measurements is then obtained by an inference algorithm applied to the language model. This technique has worked well for the speech community, and it is natural to try to suborn the model to deal with sign language.

Human gestures are rather like human sounds — there is typically a sequence of events, and we have measurements that result from these events, but do not determine them. While there is no guarantee that the rather stiff conditional independence assumptions underlying hidden Markov models are actually true for

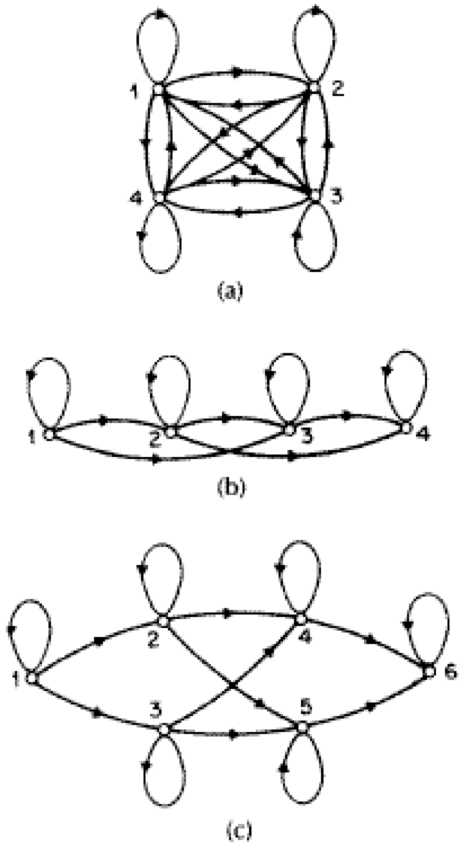


Figure 21.15. A variety of HMM topologies have proven useful in practice. At the **top**, we show an ergodic or fully connected model. In the **middle**, we show a four state left-right model. This model has been constrained so that $p_{ij} = 0$ for $j > i + 2$ as well. At the **bottom**, we show a six state parallel path left-right model. This model has, in essence, two parallel left-right models with the option of switching between them. *figure from A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, L. Rabiner, p.266, in the fervent hope of receiving permission*

human gestures, the model is certainly worth trying, because there is no such guarantee for speech models, and HMM's work in that application.

We will describe systems that use HMM's to interpret sign language, but you should realize that this approach will also work for formal systems of gestures. For example, if I wish to build a system that opens a window when I move my hand one way, and closes it if I move my hand another way, I could regard that set of gestures as a rather restricted sign language.

There are a number of systems that use HMM's to interpret sign language.

Missing
Figure

Figure 21.16. An independent word language model.

Generally, word models are left-right models, but constrained not to skip too many states (i.e. $p_{ij} = 0$ for $j > i + \Delta$ and Δ fairly small). Typically, there are a small number of states; each state has a self-loop, meaning that the model can stay in that state for a number of ticks; and there are transitions that skip some states, meaning that it is possible to move through the hidden states rather fast. Only two issues must now be resolved to build a system: firstly, the word models need to be connected together with some form of language model, and secondly, one must decide what to measure.

Language Models: Sentences from Words

We wish to recognise expressions in sign language, rather than isolated words, so we need to specify how words will appear. Language models are a specification of how word HMM's should be strung together to give an HMM that represents full sentences. The simplest language model has words that occur independent of the previous word. The language model can be expressed by a graph, as in figure 21.16. In this model, the state moves to a start state, then emits a word, then either stops or loops back from where another word can be emitted. It is now necessary to find an extremal path through this larger graph; the Viterbi algorithm still applies, there is just no observation corresponding to the shaded states.

More sophisticated language models are desirable — a language model for English that had words drawn independently at random would generate quite long strings of "and"'s and "the"'s — but do generate computational problems. The natural first step is to have a bigram language model, where the probability that a word is produced depends on the word that was produced previously. This generates a language model of the form of figure 21.17. More sophisticated language models involve trigrams — that is, the probability that a word is emitted depends on the last two words — or more detailed context from the preceeding sentence. The difficulty with this approach is that, for a reasonably sized vocabulary, the number of states the Viterbi algorithm has to search can be prodigious. There are a variety of techniques for pruning this search, which are beyond our scope — Jelinek's book gives one approach in chapter 5 [?], or see [?].

Missing
Figure

Figure 21.17. A bigram language model

Features and Typical Levels of Performance

There are a few current programs for sign language recognition. Starner has written several programs, using a variety of features. The simplest approach requires a user to wear a yellow glove on the right hand and a red glove on the left hand [**?**]. An alternative approach to wearing gloves — which is something of a nuisance — is to segment hands using skin colour (always assuming there is nothing else skin-coloured in the vicinity) [**?**]. Pixels of interesting colours (yellow and red, or skin) are identified in the image. Once an appropriate pixel has been found, its eight neighbours are checked to determine if their colour is acceptable, too; this process continues to obtain a blob of pixels.

Blobs admit a variety of features. The center of gravity of the blob gives two features, and the change in center of gravity from the previous frame yields another two. The area of the blob is another feature. The orientation and size of the blob can be measured by forming a matrix of second moments,

$$\begin{pmatrix} \int x^2 dxdy & \frac{1}{2}\int xydxdy \\ \frac{1}{2}\int xydxdy & \int y^2 dxdy \end{pmatrix}$$

The ratio of eigenvalues of this matrix gives an indication of the eccentricity of the blob; the largest eigenvalue is an estimate of the size along the principal direction; and the orientation of the eigenvector corresponding to this eigenvalue gives the orientation of the blob.

Starner's system works on a vocabulary of 40 words, representing a total of four parts of speech. The topology of the HMM's for the words is given and the parameters are estimated using the EM algorithm of section **??**. For both isolated word recognition tasks and for recognition using a language model that has five word sentences (words always appearing in the order `pronoun verb noun adjective pronoun`), the system displays a word accuracy of the order of 90%. Values are slightly larger or smaller, depending on the features and the task, etc.

Vogler and Metaxas have built a system that uses estimates of arm position, recovered either from a physical sensor mounted on the body or from a system of three cameras that measures arm position fairly accurately [**?**]. For a vocabulary of

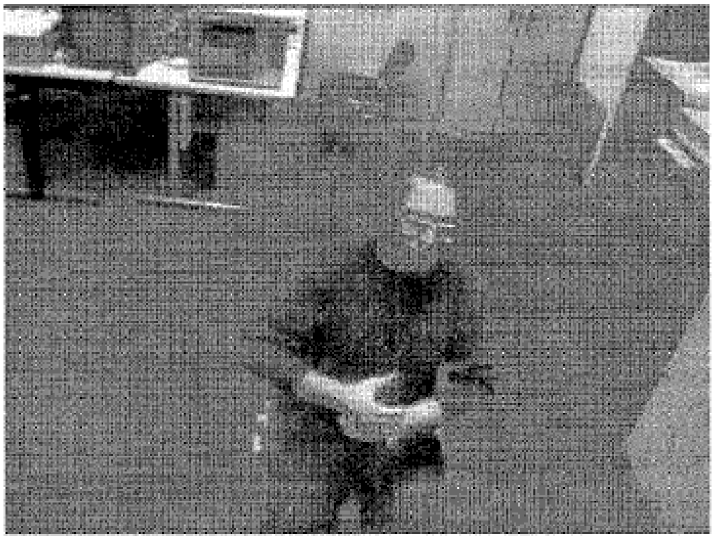


Fig. 2. View from the desk-based tracking camera. Images are analyzed at 320 × 240 resolution.

Figure 21.18. Starner and colleagues have built a sign-language recognition system that uses HMM's to yield word models. This figure shows the view of a signer presented to the system; it is obtained from a camera on the desktop. *figure from Real time American sign language recognition using desk and wearable computer based video, T. Starner, et al. p.1372, in the fervent hope of receiving permission*

53 words, and an independent word language model, they report a word recognition accuracy of the order of 90%. Isolated word recognition results of the order of 80% on a vocabulary of 95 signs were reported by Kadous [**?**].

The Future

All the results in the literature are for very small vocabularies and very simple language models. Nonetheless, HMM's seem a promising method for recognising sign language. It isn't clear that very simple features are sufficient to recognise complex signs (though it is known that very coarse scale movies of sign language are still quite intelligible [**?**]). One possible direction for progress is to use features that give a better estimate of what the fingers are doing.

Good language models — and appropriate inference algorithms — are at the core of the success of modern speech recognition systems. These models are typically obtained by measuring the frequency with which a word occurs in some context — for example, trigram frequencies. Building these models requires tremendous quantities of data, because we need an accurate estimate of the relative frequencies of quite infrequent events. For example, as Jelinek points out ([**?**], p. 75) it typically takes a body of 640, 000 words to assemble a set of 15, 000 different words. This means that some words are used very often, and that measuring word frequencies on a small body of data is extremely dangerous. Both the speech recognition community and the natural language community have a lot of experience with these difficulties, and a variety of sophisticated tricks for dealing with them (e.g. [**?**; **?**; **?**]). Future research in sign language recognition will involve learning these tricks and transferring them to the vision domain.

21.6 Application: Finding People with Hidden Markov Models

It is fairly easy to see how a hidden Markov model is a reasonable choice for recognising sign language — signs occur in a constrained random order, and generate noisy measurements. There are other applications that are not as obvious. The important property of a hidden Markov model is not the temporal sequence; it is conditional independence — that X_{i+1} is independent of the past, given X_i.

This sort of independence occurs in a variety of cases. By far the most important is finding people. We assume that people appear in images as dolls, consisting of nine body segments (upper and lower left and right arms and legs respectively, and a torso) each of which is rectangular. In particular, we assume that the left lower arm is independent of all other segments given the left upper arm; that the left upper arm is independent of all segments given the torso; and extend these assumptions in the obvious way to include the right arm and the legs, too. This gives us a hidden Markov model. We can write the model out to emphasize this point. We write X_{lua} for the configuration of the left upper arm, etc., and have

$$\begin{aligned}
P(X_t, X_{lua}, X_{lla}, X_{rua}, X_{rla}, X_{lul}, X_{lll}, X_{rul}, X_{rll}) = \; & P(X_t)P(X_{lua}|X_t) \times \\
& P(X_{lla}|X_{lua}) \times \\
& P(X_{rua}|X_t) \times \\
& P(X_{rla}|X_{rua}) \times \\
& P(X_{lul}|X_t) \times \\
& P(X_{lll}|X_{lul}) \times \\
& P(X_{rul}|X_t) \times \\
& P(X_{rll}|X_{rul})
\end{aligned}$$

which we can draw as a tree indicating the dependencies (figure 21.19).

Now assume that we observe some image measurements relating to segment configuration. For the moment, we assume that each body segment can occupy one of a finite collection of discrete states — we write the event that a particular limb is in a particular state as, say, $X_{lua} = \boldsymbol{x}_{lua}$. This event will result in a measurement, whose conditional probability can be written as $P(M = \boldsymbol{m}|X_{lua} = \boldsymbol{x}_{lua})$ etc. In particular, we will have a system of segments in the image, some of which correspond to limb segments and some of which result from noise. We assume that there are N_s segments in the image. Furthermore, we assume that the probability that a

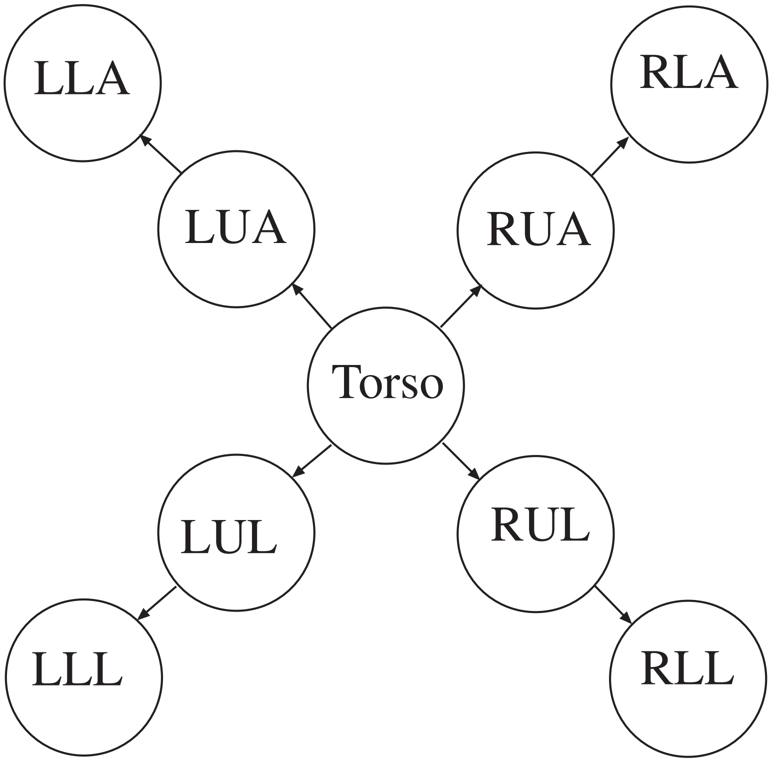


Figure 21.19. One can think of a human as forming a hidden Markov model. The tree in the figure illustrates the structure of one possible model: the torso generates various structures — arms and legs — whose properties are conditionally independent given the torso. The lower leg is conditionally independent of the rest of the model given the upper leg, etc. We can encode these independence properties by drawing a node for each variable, and a directed arc from a variable to another if the second depends directly on the first. If this drawing (which is a directed graph) is a tree, then we have a hidden Markov model. Notice that the semantics of this drawing are somewhat different from those of the drawing of figure **??**; that drawing showed the possible state transitions and their probabilities, whereas this shows the variable dependencies.

segment arises from noise is independent of anything we can measure from the segment. Now for each correspondence of image segments to limb segments we can evaluate a likelihood function. We need to be able to write out the correspondence; let us write $\{i_1, \ldots, i_9\}$ as the event that image segment i_1 is the torso, i_2 is the left upper arm, through to i_9 is the right lower leg, and that all others are noise. We write $\boldsymbol{m}_{i_k}$ for the image measurements associated with the i_k'th image segment.

Assume, for the moment, that we expect all body segments to be present at every stage. We now wish to determine which choice of image segments represents the body segments of a person who is present, and what the configuration of that person is. We have a log-likelihood that looks like:

$$\begin{aligned}\log P(\{i_1,\ldots,i_9\}|X_t=\boldsymbol{x}_t,\ldots,X_{rll}=\boldsymbol{x}_{rll}) \;=\;& \log P(\boldsymbol{m}_{i_1}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{m}_{i_2}|\boldsymbol{x}_{lua})+\ldots\\ & \log P(\boldsymbol{m}_{i_9}|\boldsymbol{x}_{rll})+\\ & (N_s-9)P(\text{image segment from noise})\end{aligned}$$

Now if we write $P(X_{lua}=\boldsymbol{x}_{lua}|X_t=\boldsymbol{x}_t)$ as $P(\boldsymbol{x}_{lua}|\boldsymbol{x}_t)$, the the log of the joint probability is

$$\begin{aligned}\log P(\{i_1,\ldots,i_9\},X_t=\boldsymbol{x}_t,\ldots,X_{rll}=\boldsymbol{x}_{rll}) \;=\;& \log P(\{i_1,\ldots,i_9\}|X_t=\boldsymbol{x}_t,\ldots,X_{rll}=\boldsymbol{x}_{rll})+\\ & \log P(\boldsymbol{x}_t,\boldsymbol{x}_{lua},\boldsymbol{x}_{lla},\boldsymbol{x}_{rua},\boldsymbol{x}_{rla},\boldsymbol{x}_{lul},\boldsymbol{x}_{lll},\boldsymbol{x}_{rul},\boldsymbol{x}_{rll})\\ =\;& \log P(\{i_1,\ldots,i_9\}|X_t=\boldsymbol{x}_t,\ldots,X_{rll}=\boldsymbol{x}_{rll})+\\ & \log P(\boldsymbol{x}_t)P(\boldsymbol{x}_{lua}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{lla}|\boldsymbol{x}_{lua})+\\ & \log P(\boldsymbol{x}_{rua}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{rla}|\boldsymbol{x}_{rua})+\\ & \log P(\boldsymbol{x}_{lul}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{lll}|\boldsymbol{x}_{lul})+\\ & \log P(\boldsymbol{x}_{rul}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{rll}|\boldsymbol{x}_{rul})\\ =\;& \log P(\boldsymbol{m}_{i_1}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{m}_{i_2}|\boldsymbol{x}_{lua})+\ldots\\ & \log P(\boldsymbol{m}_{i_9}|\boldsymbol{x}_{rll})+\\ & (N_s-9)P(\text{image segment from noise})+\\ & \log P(\boldsymbol{x}_t)P(\boldsymbol{x}_{lua}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{lla}|\boldsymbol{x}_{lua})+\\ & \log P(\boldsymbol{x}_{rua}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{rla}|\boldsymbol{x}_{rua})+\\ & \log P(\boldsymbol{x}_{lul}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{lll}|\boldsymbol{x}_{lul})+\\ & \log P(\boldsymbol{x}_{rul}|\boldsymbol{x}_t)+\\ & \log P(\boldsymbol{x}_{rll}|\boldsymbol{x}_{rul})\end{aligned}$$

Now all this fits into the lattice structure — which is the core of dynamic programming — rather easily. For each body segment, we establish a column of nodes, one for each pair of the form (image segment, body segment state). Attached to each of these nodes is a term of the form $\log P(\boldsymbol{m}_{i_1}|\boldsymbol{x}_t)$, whose value we know because we know both the image segment and the body segment state represented by the node. There is a directed arc from each element of the column representing a body segment in the tree to each element in the columns representing its children. These arcs are labelled with the logs of the appropriate "transition" probabilities, for example $P(\boldsymbol{x}_{lua}|\boldsymbol{x}_t)$ (see figure 21.20). We wish to find the directed path with the largest sum of node and arc values along the path.

The model's structure means that some nodes along the path will have more than one child; this doesn't matter. The element of dynamic programming is that it is possible to process the lattice so that, at each node, we know the value of the best choice available at that node. This element is present here, too. We transfer values back up directed arcs as before, and when a node has more than one children, we sum the best choices available over the children. Figure **??** illustrates the process for a simplified lattice.

Models of this form can be used to find people in images in a fairly straightforward manner. Firstly, we need some model linking image observations to the configuration of the limb segments — i.e. a likelihood model. Felzenszwalb and Huttenlocher assume that segments have known colour patterns — typically, a mixture of skin colour and blue — and then compare the actual image colour with these patterns. This leads to a fairly satisfactory matcher (figure 21.21), with the proviso that the person's clothing be known in advance.

21.7 Frames and Probability Models

is some function of the detector responses that is invariant to choice of coordinate frame. It is often assumed that faces are frontal and subject to rotation, translation and isotropic scale only. In this case, we can obtain invariants by either working with angles and ratios of distances, or by choosing some base points which are transformed to a canonical configuration and then measuring in that frame. Currently, few recognition systems using probabilistic reasoning represent coordinate frames explicitly. All current systems proceed in terms of transformational invariants. This is almost certainly because the topic hasn't been fully explored, rather than because of any intrinisic superiority of invariant representations (which are sometimes known as **shape spaces**).

In discussing search, we avoided discussion of *where* the face is. There are a variety of ways of handling this issue, analogous to the way it is handled in chapter **??**. The options are:

- **Represent the coordinate frame explicitly:** in this case, we build a probability model for the face in a fixed model coordinate system, and then apply that model to other coordinate systems by transforming into the model frame. Of course, to do this we need to know the model frame: we discuss the difficulties that this creates in section **??**.
- **Construct a probability model that is coordinate invariant:** in this case, our joint probability model is invariant to changes in coordinate frame.

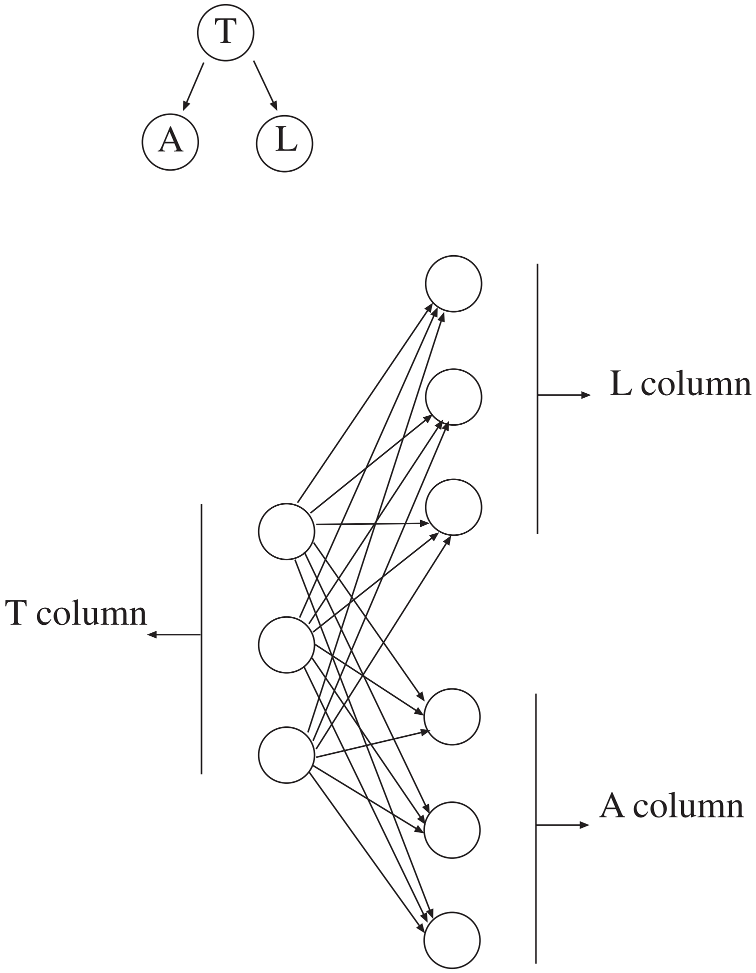


Figure 21.20. The figure shows a trellis derived from a simple tree, based around a simplified human model. You can read leg for L, etc. The elements of a column correspond to different possible correspondences between image segments, body segments and body segment configuration variables. For example, a node might represent the fact that image segment two corresponds to a torso segment at a particular position. The fact that nodes in the column marked "T" have two children each — instead of the single child in our previous example — creates no problem. For each node in this column, we can determine the best available option, and its value. This is obtained by computing the best available "A" option (and value) and the best available "L" option (and its value). In turn, this gives us the value of the "T" node. This observation means that we can use dynamic programming for any tree model.

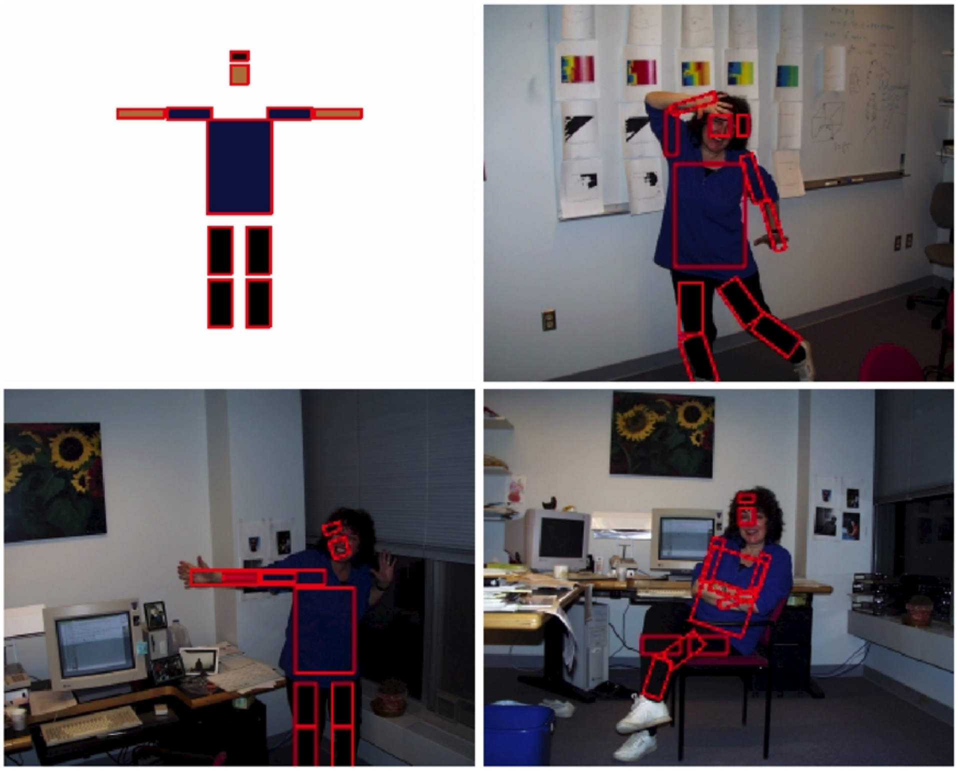


Figure 21.21. On the **top left**, a tree structured model of a person. Each segment is coloured with the image colour expected within this segment. The model attempts to find a configuration of these 11 body segments (9 limb segments, face and hair) that (a) matches these colours and (b) is configured like a person. This can be done with dynamic programming, as described in the text. The other three frames show matches obtained using the method. *figure from Efficient Matching of Pictorial Structures P. Felzenszwalb and D.P. Huttenlocher p.8, in the fervent hope of receiving permission*

> There are a variety of ways of doing this: we could build a joint probability model that was a function of the invariants of the configuration, an approach illustrated in section **??**; in section **??**, we discuss a voting technique that is analogous to geometric hashing (section **??**).

21.7.1 Representing Coordinate Frames Explicitly in a Probability Model

We will use face-finding as a running example, to illustrate the process. Assume that the feature finders report assorted information about various features in various positions. The least a detector can report is the position of its response: but it might report more, including (say) the orientation and scale of the feature. We need a joint probability distribution that represents the relative configurations that are consistent with the presence of a face. We could write this joint as

$$P(\text{feature 1 detector responds } \boldsymbol{x}_1, \ldots, \text{feature } k \text{ detector responds } \boldsymbol{x}_k|\text{object})$$

For example, assume that a face contains four features: a right eye, a left eye, a mouth and a nose. We have detectors that identify possible positions for each of these features, and write the position of a detector response as $\boldsymbol{x}_{\text{left eye}}$ (or, for brevity, $\boldsymbol{x}_{\text{le}}$).

Generally, the absolute position of the features isn't important; it is the *relative configuration* that is important. We have chosen to interpret "relative configuration" as "configuration within some model coordinate frame". This means that we need to consider the posterior probability that a face is present *and that it has been subject to a particular transformation* (or, equivalently, that a face is present, and its coordinate frame is known). For example, if we are interested in frontal views of faces, it is reasonable to assume that the faces are subject only to in-plane rotations, translations and scalings. This means that a coordinate frame is given by an origin and an angle; we write the frame as $\boldsymbol{F} = \{\boldsymbol{x}, \theta\}$, where $\boldsymbol{x}$ is the position of the origin. We would then be considering

$$P(\text{face in frame } \boldsymbol{F}|\boldsymbol{x}_{\text{le}} = \boldsymbol{X}_1, \boldsymbol{x}_{\text{re}} = \boldsymbol{X}_2, \boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3, \boldsymbol{x}_{\text{n}} = \boldsymbol{X}_4)$$

Bayes' rule yields

$$\begin{aligned} P(\text{face in frame } \boldsymbol{F}|\boldsymbol{x}_{\text{le}} = \boldsymbol{X}_1, \boldsymbol{x}_{\text{re}} = \boldsymbol{X}_2, \boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3, \boldsymbol{x}_{\text{n}} = \boldsymbol{X}_4) \propto \\ P(\boldsymbol{x}_{\text{le}} = \boldsymbol{X}_1, \boldsymbol{x}_{\text{re}} = \boldsymbol{X}_2, \boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3, \boldsymbol{x}_{\text{n}} = \boldsymbol{X}_4|\text{face in frame } \boldsymbol{F})P(\text{face in frame } \boldsymbol{F}) \end{aligned}$$

For the vast majority of vision problems, the fact that a face is present is going to be independent of its position, orientation and scale, etc., so that

$$P(\text{face in frame } \boldsymbol{F}) = P(\text{face})P(\text{frame } \boldsymbol{F})$$

Furthermore, for the vast majority of vision problems the coordinate frame doesn't affect the extent to which the configuration of features looks like a face, as long as the transformation doesn't result in features being placed outside the image. This means that, for any admissible transformation $\mathcal{T}$ — which might be any rotation, and some subset of translations and scalings — and for positions $\boldsymbol{X}_i$ such that both $\boldsymbol{X}_i \in U_i$ and $\mathcal{T}(\boldsymbol{X}_i) \in U_i$, we must have

$$\begin{aligned} P(\boldsymbol{x}_{\text{le}} = \boldsymbol{X}_1, \boldsymbol{x}_{\text{re}} = \boldsymbol{X}_2, \boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3, \boldsymbol{x}_{\text{n}} = \boldsymbol{X}_4|\text{face in frame } \boldsymbol{F}) \\ = P(\boldsymbol{x}_{\text{le}} = \mathcal{T}(\boldsymbol{X}_1), \boldsymbol{x}_{\text{re}} = \mathcal{T}(\boldsymbol{X}_2), \boldsymbol{x}_{\text{m}} = \mathcal{T}(\boldsymbol{X}_3), \boldsymbol{x}_{\text{n}} = \mathcal{T}(\boldsymbol{X}_4)|\text{face in frame } \mathcal{T}(\boldsymbol{F})) \end{aligned}$$

(The constraint on the positions is to ensure that this property only applies when, say, points in the image transform to other points in the image; if you're not sure about it why this is essential, you can ignore the point).

One way to build a model with this property is to construct a model of the way faces vary within a fixed world coordinate frame, that is

$$P(\boldsymbol{x}_{\mathrm{le}} = \boldsymbol{U}_1, \boldsymbol{x}_{\mathrm{re}} = \boldsymbol{U}_2, \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{U}_3, \boldsymbol{x}_{\mathrm{n}} = \boldsymbol{U}_4 | \text{face in frame } \boldsymbol{W})$$

Now to determine the probability model for a set of features in some other frame, we transform them to the world coordinate frame and evaluate the model there. This means that

$$\begin{aligned} &P(\boldsymbol{x}_{\mathrm{le}} = \boldsymbol{X}_1, \boldsymbol{x}_{\mathrm{re}} = \boldsymbol{X}_2, \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3, \boldsymbol{x}_{\mathrm{n}} = \boldsymbol{X}_4 | \text{face in frame } \boldsymbol{G}) = \\ &\quad P(\boldsymbol{x}_{\mathrm{le}} = \boldsymbol{U}_1, \boldsymbol{x}_{\mathrm{re}} = \boldsymbol{U}_2, \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{U}_3, \boldsymbol{x}_{\mathrm{n}} = \boldsymbol{U}_4 | \text{face in frame } \boldsymbol{W}) \end{aligned}$$

where $\mathcal{T}$ is the transformation that takes objects in frame $\boldsymbol{G}$ to objects in frame $\boldsymbol{W}$ and $\boldsymbol{U}_i = \mathcal{T}(\boldsymbol{X}_i)$.

21.7.2 Using a Probability Model to Predict Feature Positions

We can use a probability model to guide a search amongst the detector responses. To do this, we need to be able to predict conditional distributions on the position of the other features given a detector response. In the simplest case, we assume that a detector response gives only a position (more complex detectors might report a position, orientation and scale, say). In particular, we assume that, for example, the mouth detector responds when

$$P(\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3 | \text{image block centered at } \boldsymbol{X})$$

is larger than some threshold. Now assume that we have a mouth detector response at $\boldsymbol{X}_3$. We should like some guidance as to where to look for (say) the nose. This means we must look at

$$P(\boldsymbol{x}_{\mathrm{n}} | \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3)$$

which is

$$\int P(\boldsymbol{x}_{\mathrm{n}} | \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3, \mathrm{f}, \boldsymbol{F}) P(\mathrm{f}, \boldsymbol{F} | \boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3) d\boldsymbol{F}$$

To interpret this expression, we need to think more carefully about about coordinate frames. We will do two cases: models that are subject to plane translation and rotation; and models that are subject to plane translation, rotation and scaling.

Translation and Rotation

A translation and a rotation can be expressed using a point — for the origin — and an angle. We use the notation $\boldsymbol{M}$ for the world frame of the model (i.e. $\boldsymbol{M} = \{(0,0), 0\}$). We write coordinates in this frame as $\boldsymbol{U}_i$ and in some other frame as $\boldsymbol{X}_i$.

Recall that we derived $P(\boldsymbol{u}_{\mathrm{m}} | \mathrm{f}, \boldsymbol{F})$ from $P(\boldsymbol{x}_{\mathrm{m}} | \mathrm{f}, \boldsymbol{M})$ by transforming the coordinates to the world frame of the model, and then evaluating there. Generally, we expect that $P(\boldsymbol{u}_{\mathrm{m}}|\mathrm{f},\, \boldsymbol{M})$ is peaked quite tightly at some point (where the mouth occurs most often), because the components of faces don't move around all that much. One way to follow this discussion on first reading is to consider what happens when there is no variation in the position of the mouth — this means that if we know the coordinate frame, we know exactly where the mouth is. But it doesn't mean that, if we know where the mouth is, we know where the coordinate frame is.

Now we have that

$$P(\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3) \propto P(\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3|\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\})P(\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\})$$

We assume that no frame that lies in the image is advantaged over any other, which means that $P(\mathrm{f},\, \boldsymbol{F})$ is nearly uniform for all frames that lie within the image. However, since the mouth detector reports only the position of the mouth, we can't determine the orientation of the coordinate frame. We can see this by writing out the probabilities:

$$\begin{aligned} P(\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3) &\propto P(\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3|\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\})P(\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\}) \\ &= P(\boldsymbol{u}_{\mathrm{m}} = \mathcal{R}_\theta^{-1}(\boldsymbol{X}_3 - \boldsymbol{x})|\mathrm{f},\, \boldsymbol{M} = \{(0,0), 0\})P(\mathrm{f},\, \boldsymbol{F} = \{\boldsymbol{x}, \theta\}) \end{aligned}$$

Now there are many solutions in $\{\boldsymbol{x}, \theta\}$ to the equation $\mathcal{R}_\theta^{-1}(\boldsymbol{X}_3 - \boldsymbol{x}) = \boldsymbol{U}$, where $\boldsymbol{X}_3$ and $\boldsymbol{U}$ are fixed — in fact, there is a one parameter family of solutions. This means that there is a one parameter family of frames that have the same value of $P(\mathrm{f},\, \boldsymbol{F}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3)$.

In our example, the position of the mouth was known exactly given the frame. The ambiguity described here corresponds to the (geometrically obvious) point that if we know where the mouth is, we know how far away the origin of the coordinate frame is, but not *where* it is — there is a one-parameter family of possible origins, each with its own fixed orientation. The advantage of phrasing the whole process probabilistically is that we can deal with cases where the position of, say, the mouth is not exactly known given the frame.

Hence, we expect that

$$\int P(\boldsymbol{x}_{\mathrm{n}}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3, \mathrm{f},\, \boldsymbol{F})P(\mathrm{f},\, \boldsymbol{F}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3)d\boldsymbol{F}$$

will be peaked along some curve. Again, if the position of the mouth and nose were known exactly given the frame, we would know the distance from mouth to nose, and so knowing the position of the mouth would give us that the nose must lie on a circle centered at the mouth.

Translation, Rotation and Scaling

In the case of translation, rotation and scaling, the development follows as above, but with one important difference: we don't expect to be able to extract much from

$$\int P(\boldsymbol{x}_{\mathrm{n}}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3, \mathrm{f},\, \boldsymbol{F})P(\mathrm{f},\, \boldsymbol{F}|\boldsymbol{x}_{\mathrm{m}} = \boldsymbol{X}_3)d\boldsymbol{F}$$

(in fact, we'd be lucky if it wasn't very close to being uniform). This follows from a fairly simple geometric argument. Consider the case where the position of each feature is known exactly once the frame is known. If the coordinate system is subject to translation, rotation and scaling, then knowing the position of one feature does not constrain the frame at all — the origin could be any distance away at any orientation. We need at least two features to obtain any constraint on the coordinate system.

Detectors that Report more than Position

Now assume that we have detectors that report, say, the position, orientation and scale of a feature; then we are interested in the constraints that this yields on the position, orientation and scale of other features. Again, the issue to study is such conditional densities as

$$\int P(\boldsymbol{x}_{\text{n}}|\boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3, \text{f}, \boldsymbol{F})P(\text{f}, \boldsymbol{F}|\boldsymbol{x}_{\text{m}} = \boldsymbol{X}_3)d\boldsymbol{F}$$

where $\boldsymbol{X}_3$ encodes position, orientation and scale. Detectors like this are extremely useful, because *even one match can constrain the frame quite tightly.* For example, in the case of faces subject to rotation, translation and scale, reporting the position, orientation and scale of, say, the mouth should yield a tight constraint on the frame. In turn, this constrains the position, orientation and scale of the eyes and nose quite tightly.

21.7.3 Building Probability Models that are Frame-Invariant

Recall that we built an explicit representation of the coordinate frame into the probability model because we wanted to represent relationships that are unaffected by coordinate changes (rather than absolute positions, orientations, etc.). The advantages of working in this explicit coordinate frame are that it is quite easy to build models — we site a coordinate frame on a very large number of examples, and then look at the variation in position, etc. with respect to that coordinate frame — and that it is quite easy to extend familiar reasoning about correspondence to apply to this case.

Another familiar form of reasoning should be the use of invariants: instead of representing a coordinate frame in which to compute relations, we could simply code the points in a way that is invariant to transformations. This would involve building a probability model that is invariant to changes of frame. In principle, this is quite easy to do — we simply ensure that the probability model is a function only of transformational invariants of the set of detector responses.

For example, if we were interested in face finding under rotation and translation, our probability model would have to have the form

$$P(\boldsymbol{x}_{re}, \boldsymbol{x}_{le}, \boldsymbol{x}_m, \boldsymbol{x}_n|\text{face}) = f(d_{\text{re-le}}, d_{\text{re-m}}, \ldots, d_{\text{m-n}})$$

(where $d_{\text{re-le}}$ is the distance from right eye to left eye). It is stretching to call this a probability model, because there are some marginal probabilities we *cannot evaluate*. For example $P(\boldsymbol{x}_{re}|\text{face})$ isn't meaningful, because we can't talk about the distance between one point and itself. An important consequence of this face is that to do inference about the position of points, *we need to supply enough information to determine a frame in the first place.*

This means that we *can* talk about, say $P(\boldsymbol{x}_{re}|\boldsymbol{x}_{le} = \boldsymbol{X}_1, \text{face})$ but not about $P(\boldsymbol{x}_{re}|\text{face})$. Similarly, if we were working up to rotation, translation and scale, we would be able to talk about $P(\boldsymbol{x}_{re}|\boldsymbol{x}_{le} = \boldsymbol{X}_1, \boldsymbol{x}_m = \boldsymbol{X}_2, \text{face})$ but not about $P(\boldsymbol{x}_{re}|\boldsymbol{x}_{le} = \boldsymbol{X}_1, \text{face})$. This is because we have to have enough information to compute invariants before we can evaluate the probability model.

This generates a small technical muddle with predicting point positions. In particular, not every set of distances corresponds to a configuration of points: this means that evaluating $P(\boldsymbol{x}_{re}|\boldsymbol{x}_{le} = \boldsymbol{X}_1, \text{face})$ requires a little care (exercises). Apart from this issue, everything discussed in section **??** goes through: we can determine when to stop growing a group, where to look for a new feature, and what to do next with a model of this form.

21.7.4 Example: Finding Faces Using Frame Invariance

We are not aware of any recognition system that uses probabilistic reasoning and represents coordinate frames explicitly. All current systems proceed in terms of transformational invariants. This is almost certainly because the topic hasn't been fully explored, rather than because of any intrinisic superiority of invariant representations (which are sometimes known as **shape spaces**).

Perona and colleagues have built a series of face finders that use various forms of frame invariant representation. Each follows broadly the line of incremental search amongst assemblies, identifying assemblies that should be expanded. None prunes the search. Typical outputs for their feature detectors are shown in figure 21.22. They compute invariants by using reference points (which set up a canonical coordinate system, allowing invariants to be computed). As figure 21.22 indicates, the choice of reference points significantly affects the variance of the coordinates computed with respect to those reference points. This is because the relative configuration of the eyes is more stable than that of, say, the left eye and the lip.

Figure 21.23 indicates how the detectors respond to a typical face image. All these responses are searched; there is no pruning, and instead of classifying hypotheses, assemblies are ranked. The best matching assembly is a fair guide to the presence and position of a face (figure 21.24).

21.8 Conclusions

Viterbi allows pruning.

Viterbi is a consequence of conditional independence properties in a model

Other transformations are hard

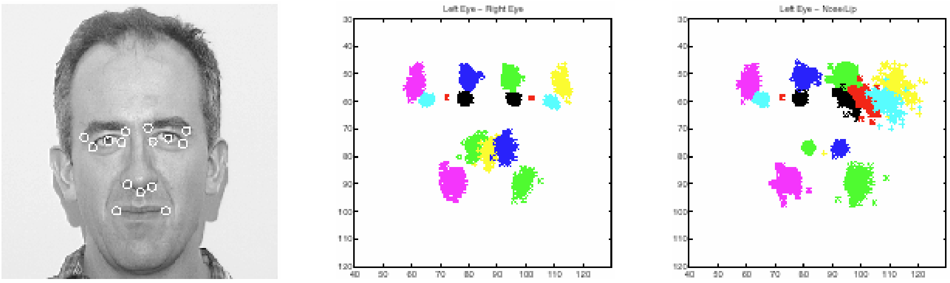


Figure 21.22. Perona's feature detector responses

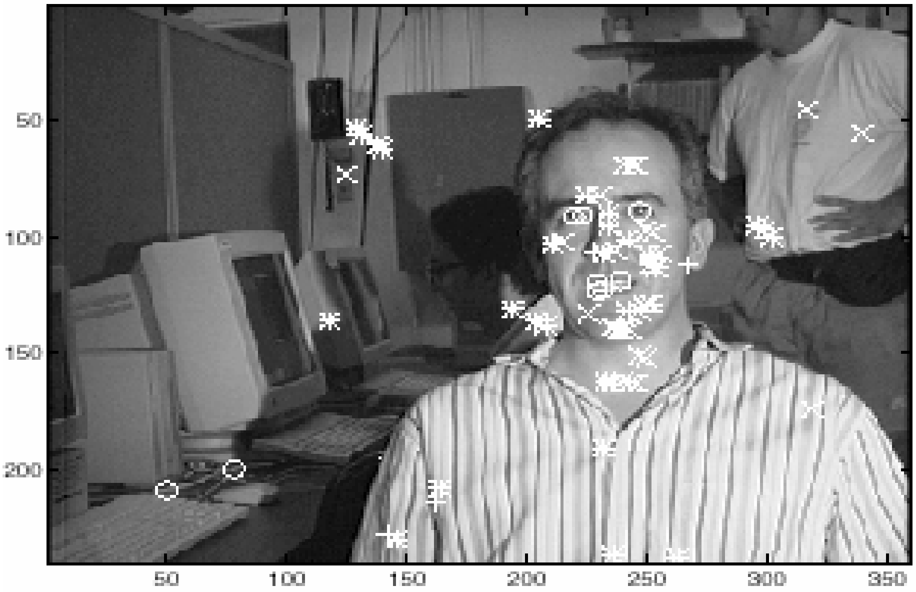

50
100
150
200
50
100
150
200
250
300
350

Figure 21.23. Perona's feature detector outputs

Models that allow efficient inference are essential

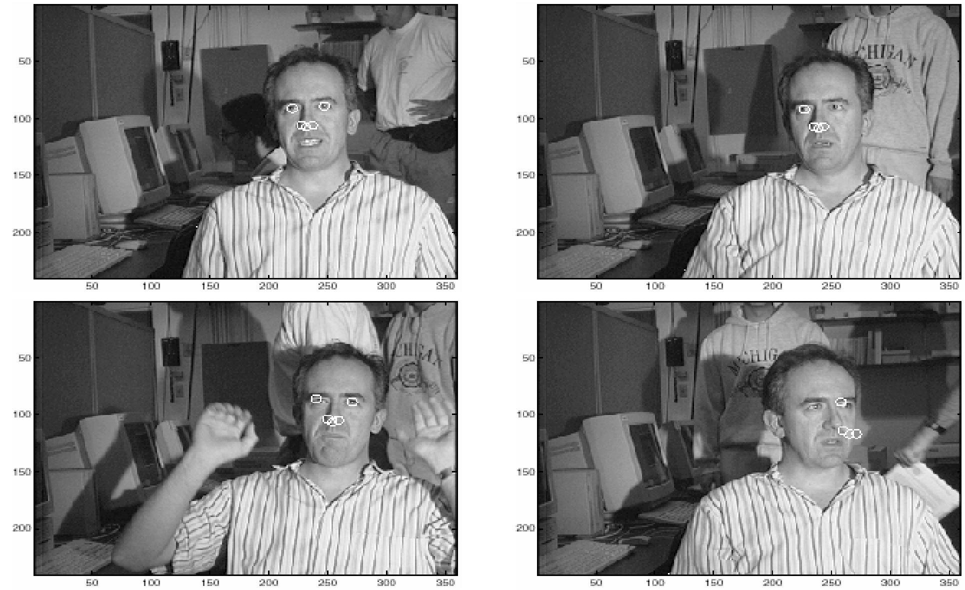

50
100
150
200
50
100
150
200
250
300
350
50
100
150
200
50
100
150
200
250
300
350
50
100
150
200
50
100
150
200
250
300
350
50
100
150
200
50
100
150
200
250
300
350

Figure 21.24. Perona's matcher

Chapter 22

ASPECT GRAPHS

This chapter addresses again the problem of characterizing the set of all pictures of an object. Unlike the quantitative techniques proposed in Chapter 20, however, the approach presented below is purely qualitative, and it relies on the mathematical tools of differential geometry and singularity theory to group images that share the same topological structure into equivalence classes. A common theme in this part of the book is that elucidating the relationship between three-dimensional shape, viewpoint, illumination and image structure is a prerequisite to developing effective recognition techniques. This is the motivation for this new twist in our study.

To illustrate the general idea, let us first pause in Flatland, a world where tiny gnats roam a two-dimensional landscape, and let us consider a planar object bounded by a smooth closed curve and observed by a perspective camera. The "occluding contour" defined in Chapter 4 consists here of the points where rays of light issued from the optical center graze the curve, and the image "contour points" are formed by the intersection of these rays with the linear retina of the camera. Objects in Flatland may be transparent or opaque; in the latter case, some of the contour rays may of course intersect (transversally) their surface before they reach the camera, in which case the corresponding contour points will not appear in the image. Either way, small camera motions will alter the position of the contour points, but will not change (in general) their number or their ordering along the retina (Figure 22.1(a)).

For certain viewpoints, however, a small movement of the camera will cause a dramatic change in image structure. For example, close-by contour points merge when the eye crosses a line that is bitangent to the outline of a transparent object, then separate again as their order reverses (Figure 22.1(b)). For an opaque object, one of the contour points becomes occluded as the eye traverses the bitangent, so the view also changes radically there, but in a different way: the order reversal is replaced by the disappearance (or appearance) of a contour point (Figure 22.1(c)). Likewise, a pair of contour points appears or disappears when the eye crosses the tangent line at a curve inflection of a transparent object (Figure 22.1(d)). For opaque objects, only one of these points is visible, the second one being hidden by the object itself (Figure 22.1(e)).

These changes in image structure are called *visual events*, and they are associated

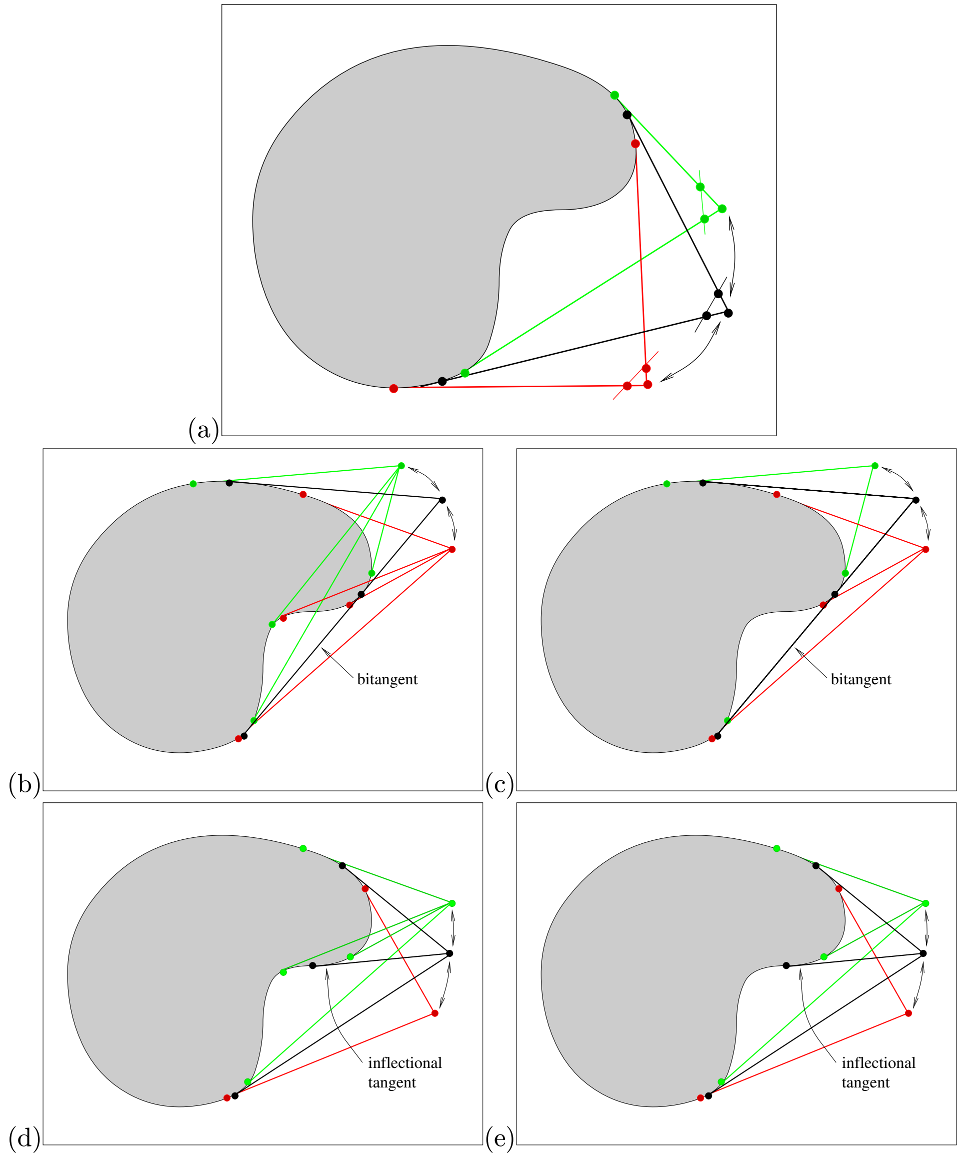


Figure 22.1. Imaging in Flatland: three perspective cameras with close-by viewpoints observe a two-dimensional solid; their one-dimensional retinas are shown in (a) as short line segments and are omitted in the rest of the figure to limit clutter. In typical views such as the one shown in (a), the order and number of contour points (or equivalently, of the associated projection rays) are the same for all three cameras, indicating a smooth mapping from viewpoint to image structure. But exceptional views induce catastrophic image changes, namely (b) a modification of the ordering of contour points along the retina, or (c)-(e) a change in the number of these points. See text for details.

with certain tangents to the boundary of the observed object (namely, bitangents and inflectional tangents). Are these events the only ones possible? Two bitangents may of course intersect; an inflectional tangent and a bitangent may intersect too, etc., yielding more complex changes in appearance. Tritangent lines and inflectional bitangents are a priori possible too. It is clear, however, that a tritangent would not survive any small deformation of the associated curve. In contrast, a bitangent or the intersection of two inflectional tangents would merely change position (Figure 22.2). In the rest of this chapter, we will restrict our attention to *generic* curves and surfaces, whose features are not affected by small deformations. Genericity is a mathematical concept related to the intuitive notion of "general position" and the topological notions of openness, density, and transversality. Its formal definition is rather technical [**?**] and would be out of place here; let us just note, following Konderink [**?**], that although the genericity assumption rules out certain simple geometric figures (lines, planes, etc.), the boundaries of all *real* objects are generic.

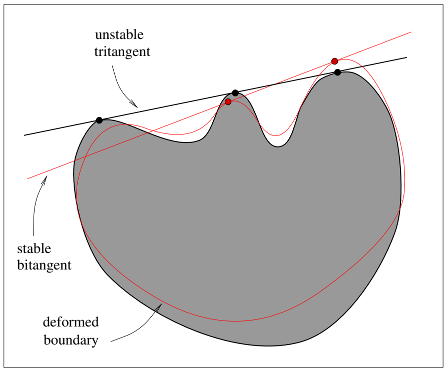


Figure 22.2. Exceptional and generic curves: unlike bitangents, tritangents are not stable under small curve deformations. It is easy to show that inflectional tangents and transversal intersections of these with bitangents or with each other are stable as well.

What may be more remarkable, at least in the context of this chapter, is that restricting one's attention to generic curves and surfaces also limits the number and type of possible visual events: indeed, structural changes in contour structure can be shown to occur in the orderly fashion predicted by the branch of mathematics called *singularity theory* or *catastrophe theory* [**?**]. In Flatland, this translates to restricting the visual events of generic curves to those associated with bitangents, inflectional tangents and their intersections. As shown in the remaining sections of this chapter, the three-dimensional case will of course be a bit more complicated..

Unlike its geometry, the topological structure of the contour (in this case the number and order of contour points) depends only on the eye position and not on the position or orientation of retina (Figure 22.1): visual events correspond to certain types of contacts between curves and projection rays and have nothing to do with the location of the points where these rays intersect the image plane (or line in this case). Thus the set of viewpoints for a Flatland camera can be modelled by the plane of the corresponding positions of its optical center. This plane is partitioned by the visual event rays and their intersections into a number of cells where the contour structure (or *aspect*) remains the same (Figure 22.3(a)). For distant observers (orthographic projection), the view space becomes a sphere at infinity, or equivalently a unit circle of projection vectors, partitioned by the directions of the bitangent and inflectional tangent lines into a finite set of aspects (Figure 22.3(b)).

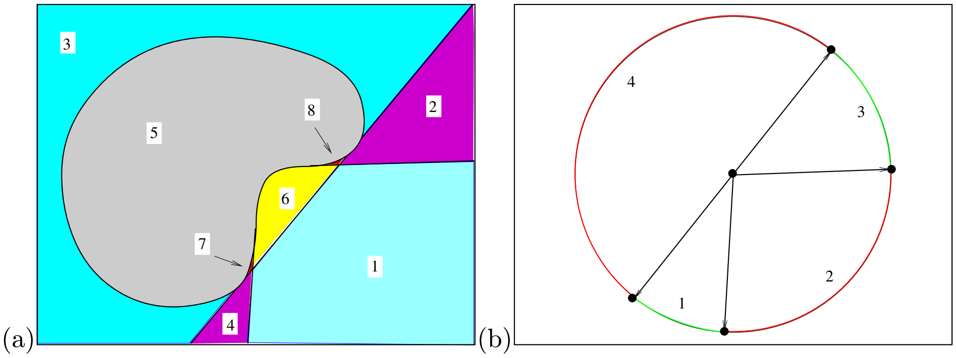


Figure 22.3. The (opaque) aspect graph in Flatland: (a) a two-dimensional object bounded by a smooth closed curve and the cells of its perspective aspect graph; note the two small regions numbered 7 and 8; (b) the corresponding orthographic aspect graph. The arcs joining pairs of adjacent aspects are not shown. Easy exercises: list these arcs; what is the transparent aspect graph of this object?

These simple examples generalize to a qualitative model of all possible views of a three-dimensional solid, the *aspect graph*, first introduced by Koenderink and Van Doorn [**?**] under the name of *visual potential*: choosing a camera model (orthographic or perspective projection) and a viewpoint determines the *aspect* of an object, i.e., the graphical structure of the observed line-drawing. The range of possible viewpoints can be partitioned into maximal regions that yield identical aspects. The change in aspect at the boundary between regions is called a visual event. The maximal regions labelled by representative aspects form the nodes of the aspect graph, whose arcs correspond to the visual event boundaries between adjacent regions.

Before telling the geometric story of aspect graphs in more detail, let us give an example. Figure 22.4(a)-(b) shows two line-drawings of a squash-shaped solid whose surface is defined as the zero set of a polynomial density function. The two curves running almost parallel to each other in Figure 22.4(a) are the parabolic curves of the squash, and they split the surface into two convex blobs separated by a saddle-shaped region. The self-intersecting curve also shown in the figure is the *flecnodal* curve that will be defined in the next section and will play a fundamental role in the definition of visual events. Figure 22.4(b) shows the *limiting bitangent* *developable* surface associated with the squash, whose rulings are the lines joining pairs of points on the squash surface that admit the same bitangent plane. The parabolic curve, the flecnodal curve and the limiting bitangent developable (as well as a couple more geometric objects that will be introduced in the next section) play a role in the formation of visual events that is analogous to the role of inflections and bitangents in the Flatland case.

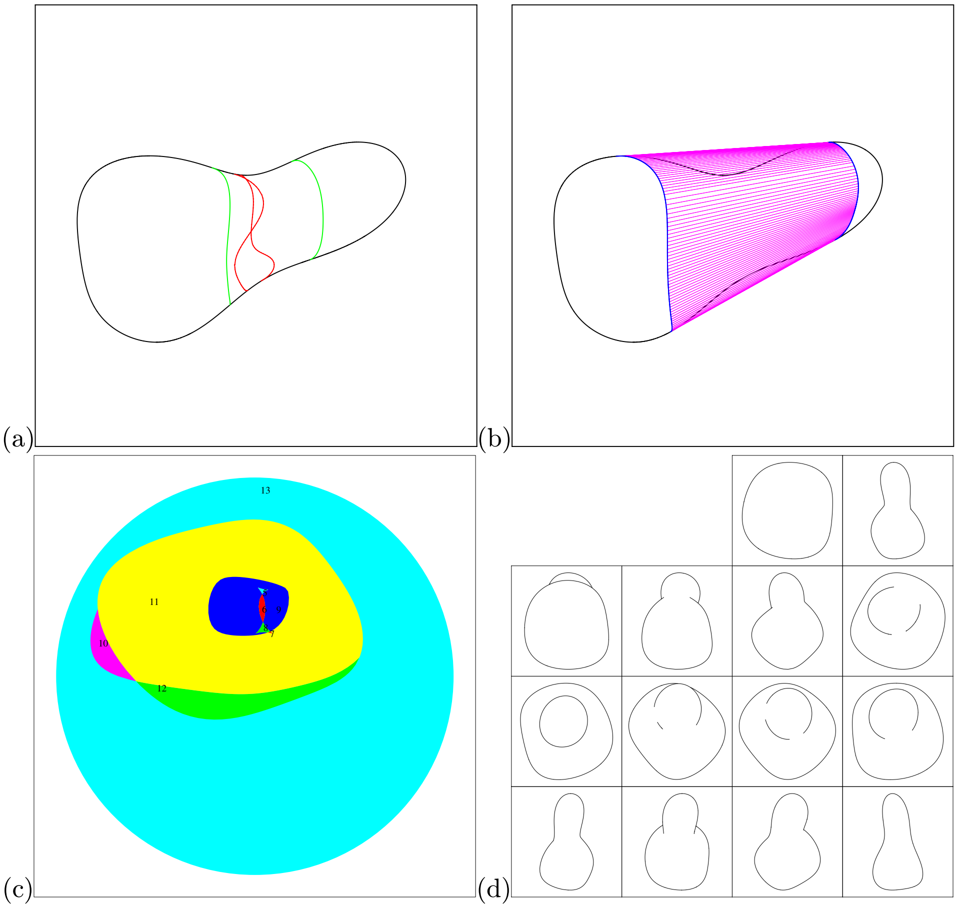


Figure 22.4. A squash-shaped object and its (opaque) aspect graph: (a) a line-drawing of the squash, with its parabolic and flecnodal curves overlaid; (b) the limiting bitangent developable of the squash; (c) the maximal regions of the viewing sphere corresponding to structurally equivalent aspects; (d) the associated aspects.

Figure 22.4(c)-(d) shows the (opaque) aspect graph of the squash assuming orthographic projection. Under this model of the imaging process, the range of possible viewpoints can be modelled as a sphere at infinity or equivalently a unit view sphere of projection directions. The aspect graph has been computed using the algorithm described in Section 22.2. The view sphere has been partitioned into maximal regions such that all points within a region see the same aspect of the object (Figure 22.4(c)). These regions are delineated by visual event curves. Representative aspects associated with each region are shown in Figure 22.4(d).

22.1 Differential Geometry and Visual Events

Inflections, cusps and T-junctions are stable features of the image contour that survive (in general) small eye movements: let us consider for example a contour inflection. As shown in Chapter 4, it is the projection of a point where the occluding contour and a parabolic curve of the associated surface intersect (normally at a non-zero angle). Any small change in viewpoint will deform the occluding contour a bit, but the two curves will still intersect transversally at a close-by point projecting onto a contour inflection.

It is natural to ask what are the (pecular) motions of the eye that will make the stable contour features appear or disappear. To answer this question, we will take in this section another look at the Gauss map and introduce the *asymptotic spherical map*, showing in the process that the boundaries of the images of a surface through these mappings determine the appearance and disappearance of inflections and cusps of its contour. This will provide us with a characterization of *local visual events*, i.e., the changes in contour structure associated with the differential geometry of the surface at these boundaries. Finally, we will consider multiple contacts between visual rays and a surface. This will lead us to the concept of *bitangent ray manifold*, and the characterization of its boundaries will allow us to understand the genesis and anihilation of T-junctions and introduce the associated *multilocal* visual events. Together, the local and multilocal events will capture the totality of the structural contour changes that determine the aspect graph.

We will assume from now on orthographic projection.

22.1.1 The Geometry of the Gauss Map

The Gauss map provides a natural setting for the study of the image contour and its inflections: indeed, we saw in Chapter 4 that the occluding contour maps onto a great circle of the unit sphere, and that the intersections of this circle with the spherical image of the parabolic set yield inflections of the contour. It is therefore clear that the contour will gain (or loose) two inflections when a camera movement causes the corresponding great circle to cross the image of the parabolic curve (Figure 22.5).

A finer understanding of the creation of pairs of inflections may be gained by taking a closer look at the geometry of the Gauss map. As shown in Chapter 4, the image of a surface on the Gauss sphere folds along the image of its parabolic set. Figure 22.6 shows an example, with a single covering of the sphere on one side of the parabolic curve and a triple covering on the other side. The easiest way of

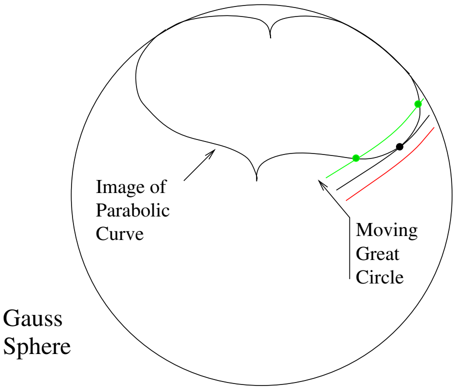


Figure 22.5. As the viewpoint changes, the great circle of the Gauss sphere associated with the occluding contour may become tangent to the spherical image of the parabolic curve. Afterwards, the circle will intersect this curve in two close-by points corresponding to two contour inflections.

thinking about the creation of such a fold is to grab (in your mind) a bit of the rubber skin of a deflated balloon, pinch it, and fold it over. As illustrated by the figure, this process will in general introduce not only a fold of the spherical image, but two cusps as well (whose preimages are aptly named *cusps of Gauss* in classical differential geometry, or *pleats* by Koenderink). Cusps and inflections of the image of the parabolic curve always come in pairs (two pairs of inflections and one pair of cusps here, but of course there may be no cusp or inflection at all, see exercises). The inflections split the fold of the Gauss map into convex and concave parts, and their preimages are called gutterpoints (Figure 22.6).

What happens to the occluding contour as the associated great circle crosses the spherical image of the parabolic set will depend on *where* the crossing happens. As shown by Figure 22.6, there are several cases: when the crossing occurs along a convex fold of the Gauss map, an isolated point appears on the spherical image of the occluding contour before exploding into a small closed loop on the unit sphere (Figure 22.6(bottom right)). If, on the other hand, the crossing occurs along a concave fold, two separate loops merge then separate with a different connectivity (Figure 22.6(top right)). These changes are of course reflected on the image contour as well, in a way that will be detailed in the next couple of sections.

The great circle associated with the occluding contour may also cross the parabolic image at a cusp of the Gauss map. Unlike crossings that occur at regular fold points, this one is in general transversal and does not impose a tangency condition on the orientation of the great circle. There is no change in the topology of the intersection (Figure 22.6(left)).

Finally, the great circle may cross the parabolic image at an inflection. The change in topology in this case is complicated [**?**; **?**] and its description would be out

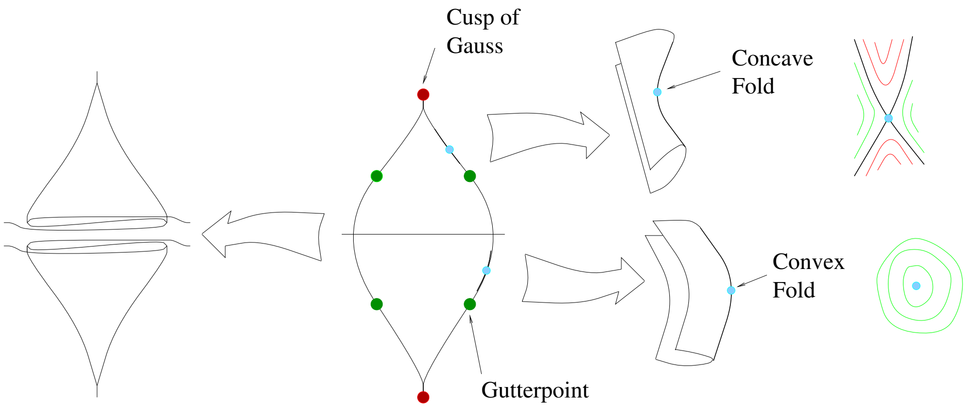


Figure 22.6. Folds and cusps of the Gauss map. The gutterpoints are the preimages of the inflections of the spherical image of the parabolic curve. To clarify the structure of the fold, it is drawn in the left and right sides of the figure as a surface folding in space. The changes in topology of the intersection between a great circle and the Gaussian image of the surface as the circle crosses the fold are illustrated in the far right of the figure.

of place here. The good news is that there is only a finite number of viewpoints for which this situation may occur (since there is only a finite number of gutterpoints on a generic surface). In contrast, the other types of fold crossings occur for infinite one-parameter families of viewpoints: this is due to the fact that the tangential crossings associated with convex or concave portions of the fold may occur anywhere along an extended curve arc drawn on the unit sphere, while the transversal crossings associated with cusps occur at isolated points but for arbitrary orientations of the great circle. We will identify the associated families of singular viewpoints in the next section.

22.1.2 Asymptotic Curves

We saw in Chapter 4 that ordinary hyperbolic points admit two distinct asymptotic tangents. More generally, the set of all asymptotic tangents on a hyperbolic patch can neatly be divided into two families such that each family admits a smooth field of integral curves, called *asymptotic curves*. Following Koenderink [**?**] we will give a color to each family of asymptotic tangents and talk about the associated *red* and *blue* asymptotic curves.

As noted by Hilbert and Cohn-Vossen [**?**, p. 204], the asymptotic curves only cover the hyperbolic part of a surface and must therefore be singular in the neighborhood of its parabolic boundary: indeed, a red asymptotic curve merges with a blue one at an ordinary parabolic point, forming a cusp and intersecting the parabolic curve at a non-zero angle (Figure 22.7(a)).[1]

---

[1] The situation is different at cusps of Gauss, where the asymptotic curves meet the parabolic

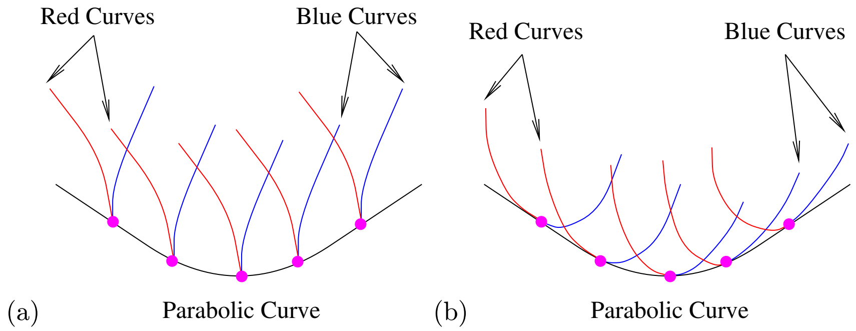


Figure 22.7. Contact of asymptotic and parabolic curves on (a) the surface and (b) the Gauss sphere.

Let us now examine the behavior of the asymptotic curves under the Gauss map. Remember from Chapter 4 that asymptotic directions are self-conjugated. This (literally) means that the derivative of the surface normal along an asymptotic curve is orthogonal to the tangent to this curve. In other words, the asymptotic curve and its spherical image have perpendicular tangents. On the other hand, all the directions of the tangent plane are conjugated to the asymptotic direction at a parabolic point, which implies that the Gaussian image of *any* surface curve passing through a parabolic point is perpendicular to the associated asymptotic direction. In particular, the Gaussian image of a parabolic curve is the *envelope* of the images of the asymptotic curves intersecting it (i.e., it is tangent everywhere to these curves, see Figure 22.7(b)).

We can now characterize the viewpoints for which a pair of inflections appears (or disappears): since the great circle associated with the occluding contour becomes tangent to the image of the parabolic curve on the Gauss sphere as they cross, the viewing direction normal to this great circle is along the associated asymptotic direction of the parabolic curve. A pair of inflections may of course also appear when the great circle crosses the image of a cusp of Gauss, or equivalently when the viewing direction crosses the tangent plane at such a point. As noted earlier, the topology of the intersection between the great circle and the image of the parabolic curve does not change in this case. Neither does the topology of the image contour, which simply gains (or looses) an *undulation*, i.e., a small concave dent in one of its convex parts, or a convex bump in one of its concave ones. The next section will show how the contour structure changes at the other types of singularities.

---

curve tangentially. This unusual behavior also occurs for planar parabolic curves of non-generic objects, e.g., the two circular parabolic curves at the top and bottom of a torus lying on its side, or more generally the parabolic lines of a solid of revolution that are associated with local extrema of the cross-section height along its axis.

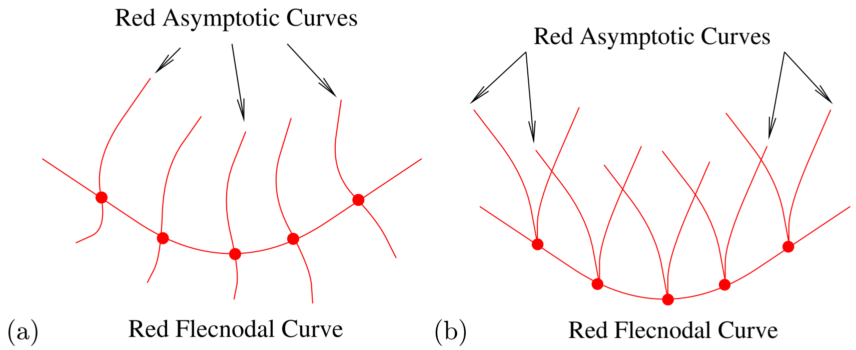
22.1.3 The Asymptotic Spherical Map

The Gauss map associates with every surface point the place where the tip of the corresponding normal pierces the unit sphere. We will now define the *asymptotic spherical map*, that associates with every (hyperbolic) point the associated asymptotic directions.

A few remarks before proceeding [**?**]: first, there is really one asymptotic spherical image for each family of asymptotic curves, and the two images may or may not overlap on the sphere. Second, the asymptotic tangents are normally not oriented, so the asymptotic spherical image of each tangent in one family consists of the two antipodal points where its direction pierces the unit sphere (or equivalently where the tangent line itself intersects the sphere at infinity). Third, it is obvious that elliptic points have no image at all, and that the unit sphere may not be fully covered by the images of the hyperbolic points. On the other hand, it may indeed be fully covered, and, at least locally, it may be covered several times by members of a single family of asymptotic directions.

Since a cusp occurs when the line of sight is along an asymptotic direction, a pair of cusps will appear (or disappear) when the viewing direction crosses one of the folds of the asymptotic spherical image (note the close analogy with the case of contour inflections and folds of the Gauss map). As was the case for the surface itself, the asymptotic spherical images of asymptotic curves must behave in a singular way near these boundaries. There are again two possibilities (the images may join the boundary tangentially, or cusp there), that do occur at two types of fold points: those associated with asymptotic directions along parabolic curves (since there is no asymptotic direction at all on the elliptic side of these curves), and those associated with an asymptotic direction at a *flecnodal* point. Flecnodal points are inflections of the projections of the asymptotic curves into their tangent plane (Figure 22.8(a)). They form curves intersecting transversally the associated asymptotic curves, and, like those, they come in two colors, depending on which asymptotic family has an inflection. Like the circular tangential image of a curve at an inflection (Chapter 4), the asymptotic spherical image of an asymptotic curve cusps at a flecnodal point (Figure 22.8(b)). It should also be noted that flecnodal curves intersect parabolic ones tangentially at cusps of Gauss.

It is clear that the contour structure will change when the line of sight crosses the parabolic or flecnodal boundaries of the asymptotic spherical image. Such a change is called a *visual event* and the associated boundaries are called *visual event curves.* Before examining in more detail the various visual events, let us note a different, but equivalent way of thinking of the associated boundaries: if we draw the singular asymptotic tangent line at each point along a parabolic or flecnodal surface we obtain *ruled surfaces* swept by the tangents. A visual event will occur whenever the line of sight crosses one of these ruled surfaces, whose intersections with the sphere at infinity are exactly the visual event curves when this sphere is identified with the unit sphere. Thinking of contour evolution in terms of these ruled surfaces has the advantages of pointing toward a generalization of visual events to



(a) (b)

Figure 22.8. Contact of asymptotic and flecnodal curves on (a) the surface and (b) the asymptotic spherical image.

perspective projection (the view will change whenever the optical center crosses them) and of allowing a clear visualization of the relationship between singular viewpoints and surface shape.

22.1.4 Local Visual Events

We are now in a position to understand how the contour structure changes at visual event boundaries. There are three *local* visual events, that are completely characterized by the local differential surface geometry: the *lip*, *beak-to-beak* and *swallowtail*. Their colourful names are inherited from Thom's catalogue of elementary catastrophes [**?**] and are closely related to the shape of the contour near the associated events.

Let us first examine the *lip* event, that occurs when the line of sight crosses the asymptotic spherical image of a "convex" parabolic point, or equivalently, the ruled surface defined by the associated asymptotic tangents (Figure 22.9(top)). We have shown earlier that the intersection between the great circle associated with the occluding contour and the Gaussian image of the surface acquires a loop during the event (Figure 22.6(bottom right)) with the creation of two inflections and two cusps on the contour. More precisely, there is no image contour before the event, with an isolated contour point appearing out of nowhere at the singularity, before exploding into a closed contour loop consisting of a pair of branches meeting at two cusps (Figure 22.9(bottom)). One of the branches always has two inflections and is formed by the projection of both elliptic and hyperbolic points, while the other one is formed by the projection of hyperbolic points only. For opaque objects, one of the branches is always occluded by the object.

The *beak-to-beak* event occurs when the line of sight crosses the asymptotic spherical image of a "concave" parabolic point, or once again, the ruled surface defined by the associated asymptotic tangents (Figure 22.10(top)). As shown ealier, the topology of the intersection between the great circle associated with the occluding

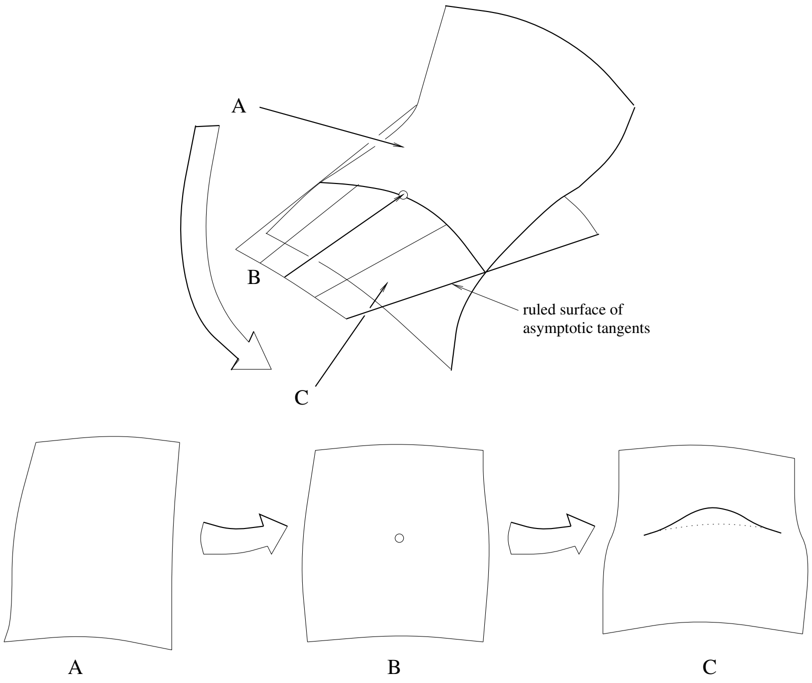


Figure 22.9. A lip event. The name is a literal translation of the French "lèvre" in Thom's catalogue of elementary catastrophes. It is of course related to the shape of the contour on the right of the figure. Here, as in latter figures, the dashed part of the contour would be invisible due to occlusion for an opaque object. In this example, the two inflections are on the visible part of the contour, the hidden part being all hyperbolic, but the situation would be reversed by taking a viewpoint along the opposite direction.

contour and the Gaussian image of the surface changes during this event, with two loops merging then splitting again with a different connectivity (Figure 22.6(top right)). In the image, two distinct portions of the contour, each having a cusp and an inflection, meet at a point in the image. Before the event, each of the branches is divided by the associated cusp into a purely hyperbolic portion and a mixed elliptic-hyperbolic arc, one of which is always occluded. After the event, two contour cusps and inflections disappear as the contour splits into two smooth arcs with a different connectivity. For opaque objects, one of these is purely elliptic while the other is purely hyperbolic, with one of the two always being occluded (Figure 22.10(bottom) –the reverse transition is of course also possible, as for all other visual events).

Finally, the *swallowtail* event occurs when the eye crosses the surface ruled by the asymptotic directions along a flecnodal curve of same color. We know that two cusps appear (or disappear) in this event. As shown in Figure 22.11(a)-(b), the intersection of the surface and its tangent plane at a flecnodal point consists

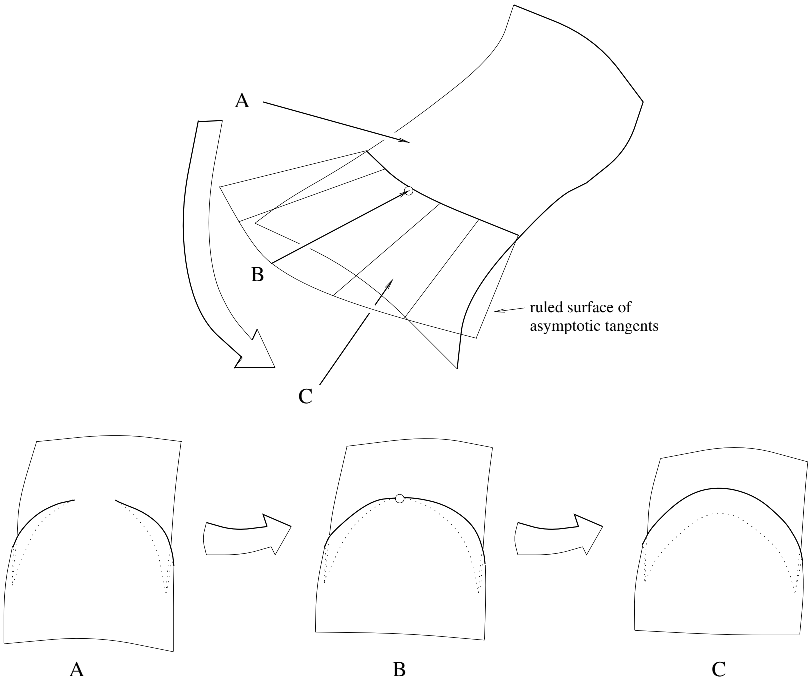


Figure 22.10. A beak-to-beak event, from the French "bec-à-bec". The name is related to the shape of the contour on the left of the figure. See text for details.

of two curves, one of which has an inflection at the point. The corresponding asymptotic tangent is of course associated with the family of asymptotic curves having an inflection there too. Unlike ordinary asymptotic rays (Figure 22.11(c)), that are blocked by the observed solid, this one intersects the solid at a single point: it "sees through it" [**?**]. This produces a sharp V on the image contour at the singularity. The contour is smooth before the transition but it acquires two cusps and a T-junction after it (Figure 22.11(bottom)). All surface points involved in the event are hyperbolic. For opaque objects, one branch of the contour ends at the T-junction and the other one ends at a cusp.

22.1.5 The Bitangent Ray Manifold

Now, remember from Chapter 4 that cusps and inflections are not the only kinds of stable contour features: T-junctions also occur over open sets of viewpoints. They form when two distinct pieces of the occluding contour project onto the same image location. The corresponding surface normals must of course be orthogonal to the *bitangent* line of sight joining the two points, but they are not (in general) parallel. That T-junctions are stable over small eye movements is intuitively clear: consider a convex point P and its tangent plane (Figure 22.12(a)). This plane intersects

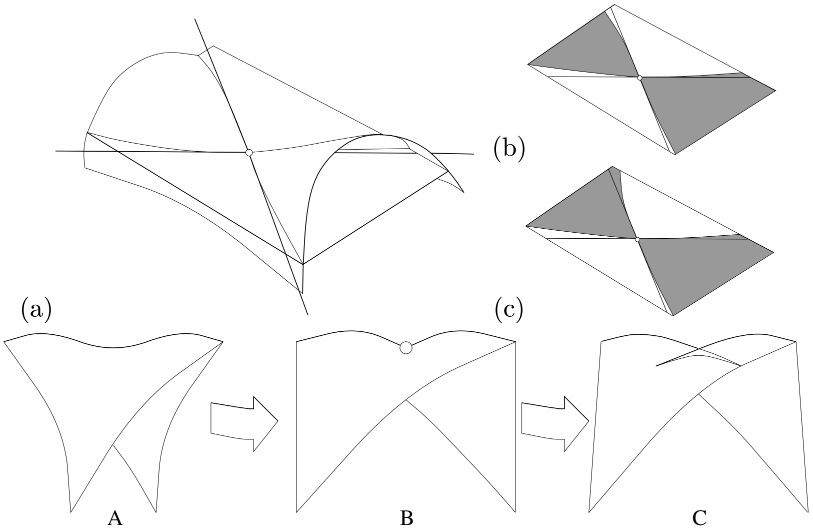


Figure 22.11. A swallowtail event (from the French “queue d’aronde”). Top: surface shape in the neighborhood of a flecnodal point (a), and comparaison of the intersection of the associated solid and its tangent plane near such a point (b) and an ordinary hyperbolic point (c). Bottom: the event itself.

(in general) the surface along a closed (but possibly empty) curve, and there will be an even number of points (P' and P'' in the figure) such that the rays drawn from P through these points are tangent to the curve. Each such tangency yields a bitangent ray and an associated T-junction. A small motion of the eye induces a small deformation of the intersection curve, but does not change (in general) the number of tangent points. Thus T-junctions are indeed stable.

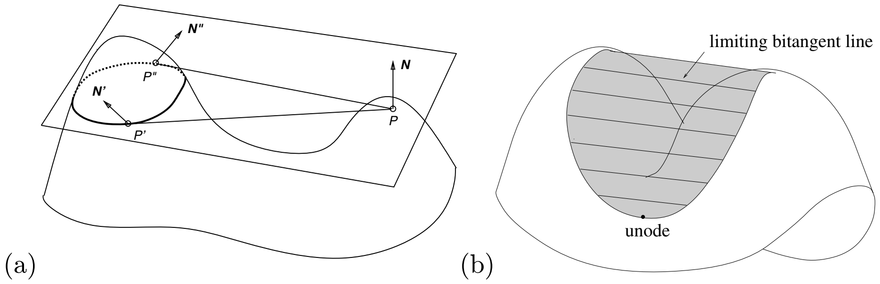


Figure 22.12. Bitangent rays: (a) the tangent plane to the surface in P intersects it along a closed curve with two bitangent rays PP' and PP'' grazing the surface along this curve; (b) the limiting bitangent developable surface ruled by the lines where a plane bitangent to a surface grazes it. Here the two curves where it touches the observed surface merge tangentially at a *unode*, a type of cusp of Gauss.

The set of all bitangent rays is called the *bitangent ray manifold* associated with the observed object, and it forms a two-dimensional surface[2] in the four-dimensional space formed by all straight lines. Since bitangents map onto T-junctions in the projection process, it is clear that these contour features will be created or destroyed at boundaries of the manifold. Since a T-junction appears or disappears during a swallowtail transition, it is also obvious that the surface ruled by the singular asymptotic tangents along a flecnodal curve must be one of these boundaries. What is not as clear is what the remaining boundaries are made of. This is the topic of the next section.

22.1.6 Multilocal Visual Events

A pair of T-junctions will appear or disappear when the line of sight crosses the boundary of the bitangent ray manifold. The corresponding change in contour structure is called a *multilocal* visual event. This section will show that there are three types of multilocal events, namely the *tangent crossing*, *cusp crossing* and *triple point*, besides the singularity associated with the crossing of a flecnodal curve that was mentioned in the previous section.

Let us first have a look at the *tangent crossing* event. An obvious boundary of the bitangent ray manifold is formed by the *limiting bitangents* (Figure 22.12(b)), that occur when the curve formed by the intersection between the tangent plane at some point and the rest of the surface shrinks to a single point, and the plane itself becomes bitangent to the surface. The limiting bitangents sweep a developable surface, called the *limiting bitangent developable.* A tangent crossing occurs when the line of sight crosses this surface (Figure 22.13(top)), with two separate pieces of contour becoming tangent to each other at the event before crossing transversally at two T-junctions (Figure 22.13(bottom)). For opaque objects, either a previously hidden part of the contour becomes visible after the transition, or (as in the figure) another contour arc disappears due to occlusion.

The bitangent ray manifold admits two other kinds of boundaries, also associated with bitangents that touch the surface along a set of curves and sweep developable surfaces: the *asymptotic bitangents*, that intersect the surface along an asymptotic direction at one of their endpoints (Figure 22.14(a)), and the *tritangents*, that graze the surface in three distinct points (Figure 22.14(b)). The corresponding visual events occur when the line of sight crosses one of the associated developable surfaces, and they also involve the appearance or disappearance of a pair of T-junctions: a *cusp crossing* occurs when a smooth piece of the image contour crosses another part of the contour at a cusp (or endpoint for an opaque object) of the latter (Figure 22.14(c)). Two T-junctions are created (or destroyed) in the process, only one of which is visible for opaque objects. A *triple point* is formed when three sep-

---

[2] Hence the name (after [?]): a manifold is a topological concept generalizing surfaces to more abstract settings; its formal definition will be omitted here. It is intuitively clear that the bitangent ray manifold is two-dimensional since there is a finite number of bitangent rays for each point of the (two-dimensional) surface being observed.

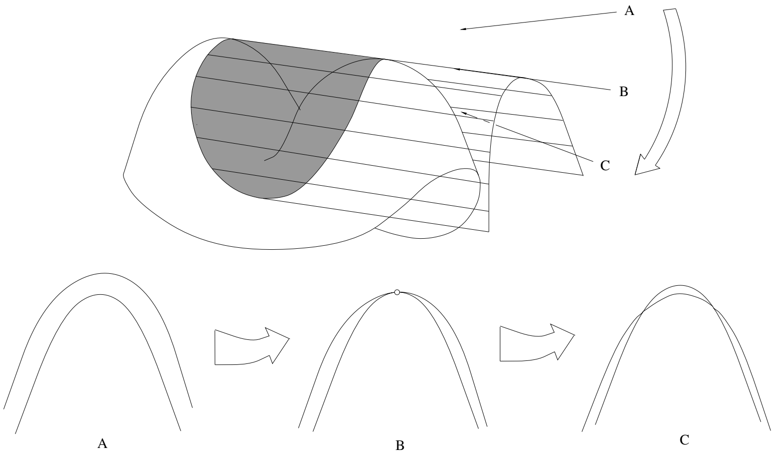


Figure 22.13. A tangent crossing event. The occlusion relationship between spatially distinct parts of the occluding contour changes when the viewpoint crosses the limiting bitangent developable surface in B.

arate pieces of the contour momentarily join at non-zero angles (Figure 22.14(d)). For transparent objects, three T-junctions merge at the singularity before separating again. For opaque objects, a contour portion disappears (or appears) after the transition, along with two T-junctions, while a third T-junction appears (or disappears).

22.1.7 Remarks

Dear reader, it is now time for a bit of apology: we must confess to having taken in this chapter a rather quick and dirty approach to the characterization of visual events: of course, it is intuitively clear that the various visual event curves (or equivalently the associated ruled surfaces) that we have discussed do indeed bound sets of viewpoints where the contour structure suddenly changes. It may even be argued based on geometric intuition that the structural changes of the contour behave as described (this is fairly clear for, say, lip or tangent crossing events, not so obvious for, say, swallowtails). But it is not clear at all that other types of events may not occur, or that the dimensionality of these events is really the same as a curve for arbitrary objects.

As before in this book, we cannot fully justify here that our catalogue of events is correct and complete. The corresponding proofs and the concepts they depend on belong to a field of mathematics that goes under various names, including differential topology, singularity theory, and catastrophe theory [**?**; **?**; **?**]; they would be out of place here and rely deeply on the assumption of genericity mentioned several times already. Let us just note a few facts of interest: first, there is a strong link between the various visual events and the types of contact that may exist between a line and

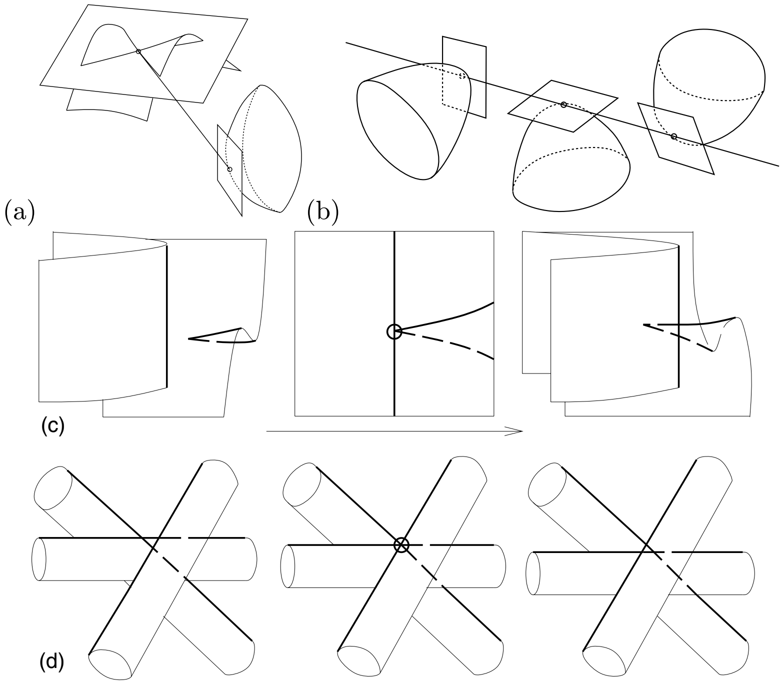


Figure 22.14. Multilocal events: (a) an asymptotic bitangent ray; (b) a tritangent ray; (c) a cusp crossing; (d) a triple point. After [**?**] (Figure Y).

a surface: a line may miss the object completely (this is called order-zero contact), intersect its surface tranversally (order-one contact) or be tangent to it (order-two contact or higher).[3] All points on a generic curved surface have an infinity of order-two tangents, spanning their tangent plane, but the two asymptotic tangents of ordinary hyperbolic points have order-three contact. Tangents of order four or higher only exist along parabolic and flecnodal curves. The order-four tangent at a flecnodal point is its asymptotic tangent with the same color, and the order-four tangent at a parabolic point is the unique asymptotic tangent there (surprise!).

Order-five contact may in fact occur at isolated points of parabolic and flecnodal curves, including the cusps of Gauss and gutterpoints mentioned earlier, but also the so-called biflecnodes and quadric points [**?**]; there are no tangents of order six on a generic surface. These special points and the associated tangents map onto singularities and intersections of the visual event curves on the view sphere, and they can be regarded as punctual visual events. The change in contour structure in their neighborhood is well understood, and they play a key role in the definition of perspective aspect graphs. Likewise, it can be shown that the multilocal events defined in the previous section correspond to the maximum order of contact that a family of bitangents and a surface may maintain along a curve, although higher-order contact may occur for isolated rulings of the associated developable surfaces. Once again, a catalogue of all possibilities is available for generic surfaces [**?**; **?**].

---

[3] The notion of order-n contact has a precise mathematical definition that will not be given here. Intuitively, it is related to the number of terms that vanish in the Taylor expansion of the surface along the line of interest. It can also be thought of as the number of (infinitesimally close) points where the line touches the surface.

22.2 Computing the Aspect Graph

Given some surface model of a solid (in the form of a polyedron, the zero set of a volumetric density, or whatever your fancy may be), the next question one may ask is how to actually construct the aspect graph of this solid. This is not a priori obvious, due in part to the deliberately descriptive approach that we have taken so far.

It is easy to rewrite the geometric definitions of parabolic points, limiting bitangent rays, etc. in terms of the derivatives (of order up to three) of the surface parameterization associated with the given model at one, two, or three surface points [**?**]. In each case, the surface curves associated with the ruled surfaces defined earlier can be characterized in $\mathbb{R}^{n+1}$ by a system of n equations in $n+1$ unknowns:

$$\begin{cases} P_1(x_0, x_1, \dots, x_n) = 0, \\ \dots \\ P_n(x_0, x_1, \dots, x_n) = 0, \end{cases} \tag{22.2.1}$$

with $1 \le n \le 8$, depending on the type of event and whether the surface is defined parametrically or implicitly. For example, in the case of a surface defined implicitly as the zero set of some density function $F(x, y, z) = 0$, we have $n = 1$ for a local event and $n = 8$ for a triple point involving three separate surface points.

Given these equations, a general approach to constructing the aspect graph of a given object involves the following steps: (1) tracing the visual event curves of the transparent object; (2) constructing the regions of the view sphere delineated by these curves; (3) eliminating the occluded events and merging the incident regions; (4) constructing the corresponding aspects. The aspect graph of a transparent solid can be constructed by using the same procedure but omitting step (3).

Here we will address the problem of constructing the aspect graph of a solid bounded by an *algebraic surface*, i.e., the zero set of a volumetric *polynomial* density:

$$S = \{(x, y, z) \in \mathbb{R}^3 | \sum_{i+j+k \le d} a_{ijk} x^i y^j z^k = 0\}.$$

Examples of algebraic surfaces include planes and quadric surfaces (i.e., ellipsoids, hyperboloids and paraboloids) as well as the zero set of higher-degree polynomial densities. Most importantly, the equations (22.2.1) defining the visual events in this case are all polynomials in the unknowns of interest. This is the key to successfully implementing the various steps of the general algorithm outlined earlier, since both numerical and symbolic computational tools are available for characterizing explicitly the solutions of multivariate polynomial equations.

22.2.1 Step 1: Tracing Visual Events

As shown in Sect. 22.1, a visual event is associated with two curves: the first one, call it Γ, is either drawn on the object surface or (in the case of multilocal events) in a higher-dimensional space. The second one, call it Δ, is drawn on the view sphere as the set of corresponding viewpoints.

The curve Γ is defined by the n equations of (22.2.1) in $\mathbb{R}^{n+1}$. This section addresses the problem of *tracing* this curve, i.e. (in our context), identifying its smooth arcs and its singularities, constructing a sample for each of these, and establishing the connectivity of the various curve components. The output of the process is a graphical description of the curve. A similar description of Δ is easily obtained by mapping the points of Γ onto the corresponding asymptotic or bitangent directions.

A simple approach to tracing Γ is the variant of the plane-sweep algorithm presented below (Figure 22.15):

(1.1) Compute all extremal points (including singular points) of Γ in some direction, say x_0.
(1.2) Compute all intersections of Γ with the hyperplanes orthogonal to the x_0 axis at the extremal points.
(1.3) For each interval of the x_0 axis delimited by these hyperplanes, intersect Γ and the hyperplane passing through the mid-point of the interval to obtain one sample for each real branch.
(1.4) March numerically from the sample points found in step (1.3) to the intersection points found in step (1.2) by predicting new points through Taylor expansion and correcting them through Newton iterations.

Step (1.1) requires the computation of the extrema of Γ in the x_0 direction. These points are characterized by a system of $n+1$ polynomial equations in $n+1$ unknowns, obtained by adding the equation $|\mathcal{J}| = 0$ to (22.2.1), where $\mathcal{J}$ denotes the Jacobian matrix $(\partial P_i/\partial x_j)$, with $i, j = 1, .., n$. They are found using the homotopy continuation method [**?**]. Steps (1.2) and (1.3) require computing the intersections of a curve with a hyperplane. Again, these points are the solutions of polynomial equations, and they are found using homotopy continuation. The curve is actually traced (in the classical sense of the term) in step (1.4), using a classical prediction/correction approach based on a Taylor expansion of the P_i's. This involves inverting the matrix $\mathcal{J}$ which is guaranteed to be non-singular on extrema-free intervals.

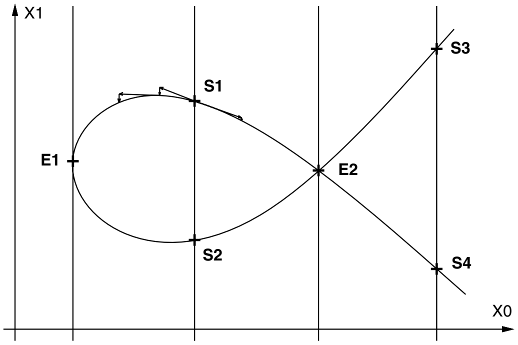


Figure 22.15. An example of curve tracing in $\mathbb{R}^2$. Hyperplanes are straight lines in this case. This curve has two extremal points $\boldsymbol{E}_1, \boldsymbol{E}_2$, and four regular branches with sample points $\boldsymbol{S}_1$ to $\boldsymbol{S}_4$. $\boldsymbol{E}_2$ is singular. Reprinted from [?], Figure X.

22.2.2 Step 2: Constructing the Regions

To construct the aspect graph regions delineated by the curves Δ on the view sphere, we refine the curve tracing algorithm into a cell decomposition algorithm whose output is a description of the regions, their boundary curves, and their adjacency relationships. Note that this refinement is only possible for curves drawn in two-dimensional spaces such as the sphere. The algorithm is divided into the following steps (Figure 22.16):

(2.1) Compute all extremal points of the curves in the x_0 direction.
(2.2) Compute all the intersection points between the curves.
(2.3) Compute all intersections of the curves with the "vertical" lines orthogonal to the x_0 axis at the extremal and intersection points.
(2.4) For each interval of the x_0 axis delimited by these lines, do the following:
  (2.4.1) Intersect the curves and the vertical line passing through the mid-point of the interval to obtain a sample point on each real branch of each curve.
  (2.4.2) Sort the sample points in increasing x_1 order.
  (2.4.3) March from the sample points to the intersection points found in step (2.3).
  (2.4.4) Two consecutive branches within an interval of x_0 and the vertical segments joining their extremities bound a region.
(2.5) For each region, a sample point can be found as the mid-point of consecutive samples found in step (2.4.1).
(2.6) Merge all regions adjacent along vertical line segments to form maximal regions. (Two regions are adjacent if they share a common boundary, i.e., a vertical line segment or a curve branch.)

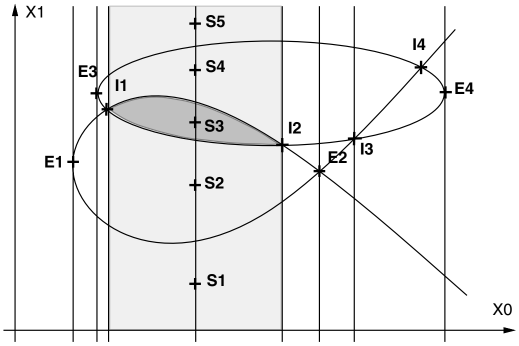


Figure 22.16. An example of cell decomposition. Two curves are shown, with their extremal points $\mathbf{E}_i$ and their intersection points $\mathbf{I}_j$. The shaded rectangle delimited by $\mathbf{I}_1$ and $\mathbf{I}_2$ is divided into five regions with sample points $\mathbf{S}_1$ to $\mathbf{S}_5$. The region corresponding to $\mathbf{S}_3$ is shown in a darker shade. Reprinted from [**?**], Figure X.

22.2.3 Remaining Steps of the Algorithm

Step 3 and 4 of the algorithms are concerned with the elimination of the occluded visual events and the construction of a sample aspect for each region. Both steps are conceptually simple.

Note that all visual events of the transparent object are found in step (1) of the algorithm, and that their intersections are found in step (2). For an opaque object, some of the events will be occluded and should be eliminated. Since the visibility of a visual event curve only changes at its singularities and intersections with other visual events, occluded events can be eliminated through ray tracing [**?**; **?**] at the sample point of each branch found in step (2.3). Regions adjacent to occluded events are readily merged.

The final step of the algorithm involves determining the contour structure of a single view for each region, first for the transparent object, then for the opaque one. Given the sample viewpoint of a region, the curve tracing algorithm described in Section 22.2.1 is applied to the image contour to construct an *image structure graph* whose nodes are the contour extrema and singularities (cusps and T-junctions), and whose arcs are the smooth branches joining them. As before, ray tracing is then used to determine whether the sample point associated with each branch is occluded or not.

22.2.4 An Example

Figure 22.17(a)-(d) shows an object in the shape of a kidney bean, its parabolic and flecnodal curves, the developable surface formed by its tangent crossings, and the corresponding visual event curves on the view sphere. This object has a hyperbolic patch within a larger convex region. The bean is bounded by a quartic (i.e., degree-four) algebraic surface whose equation is:

$$23x^4 + x^2y^2 - 37x^2y - 2xy^2 - 15x^2 - 2xy + 16y^2 + 16z^2 - x + 16y = 0.$$

As shown in Figure 22.17(d)-(f), the beak-to-beak and lip events form the hourglass pattern predicted by Koenderink [**?**]. Some of the regions are in fact extremely small, and the difference between the corresponding aspects would certainly not be discernible with a real camera having only finite resolution. Figure 22.18(a)-(b) shows the aspect graph of the transparent bean and the corresponding 25 aspects, and Figure 22.18(c)-(d) shows the aspect graph of the opaque bean and the corresponding 16 aspects.

The aspect graph shown in Figure 22.18 does not include the events corresponding to the creation or destruction of inflections: for example, the smooth aspect marked y in Figure 22.18(c)-(d) corresponds to the most common view of the bean. Its contour is convex, unlike the smooth “bean-shaped” contour our intuition seems to dictate. Fortunately, we already know that the visual boundaries corresponding to the birth and death of inflections correspond to viewpoints crossing the tangent plane at the cusps of Gauss of the surface. Therefore it is sufficient to add to the event curves the associated great circles and recompute the tessellation of the view sphere. Figure 22.19(a)-(b) shows the corresponding aspect graph of the transparent bean, with its 38 regions, and Figure 22.19(c)-(d) shows the three different aspects corresponding to the region marked y in Figure 22.18(c)-(d).

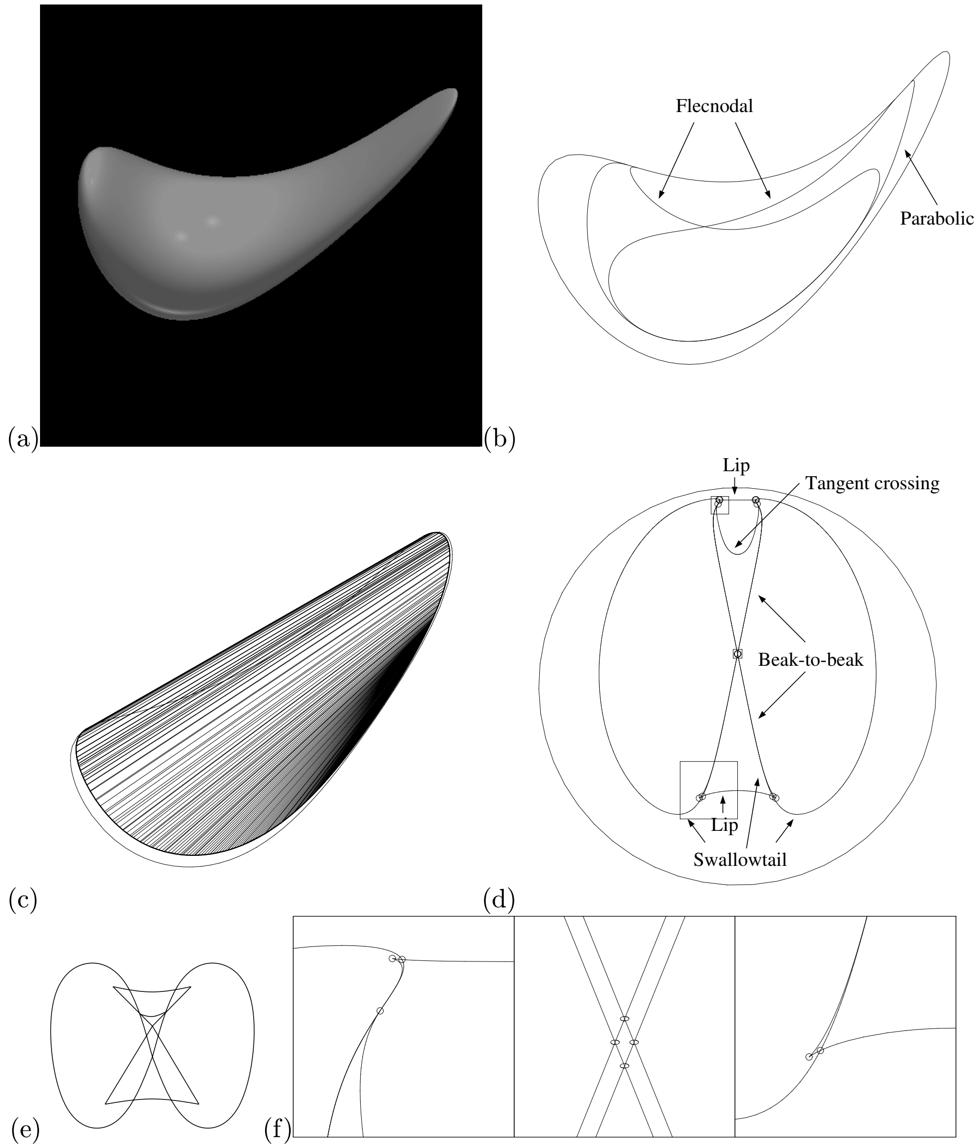

(a)
(b)
Flecnodal
Parabolic
(c)
(d)
Lip
Tangent crossing
Beak-to-beak
Lip
Swallowtail
(e)
(f)

Figure 22.17. The kidney bean and its aspect graph: (a) the bean; (b) its parabolic and flecnodal curves; (c) the developable surface corresponding to its tangent crossings; (d) the corresponding visual event curves on the view sphere; (e) the aspect graph structure predicted in [**?**; **?**; **?**]; (f) close-ups of the square regions in (d).

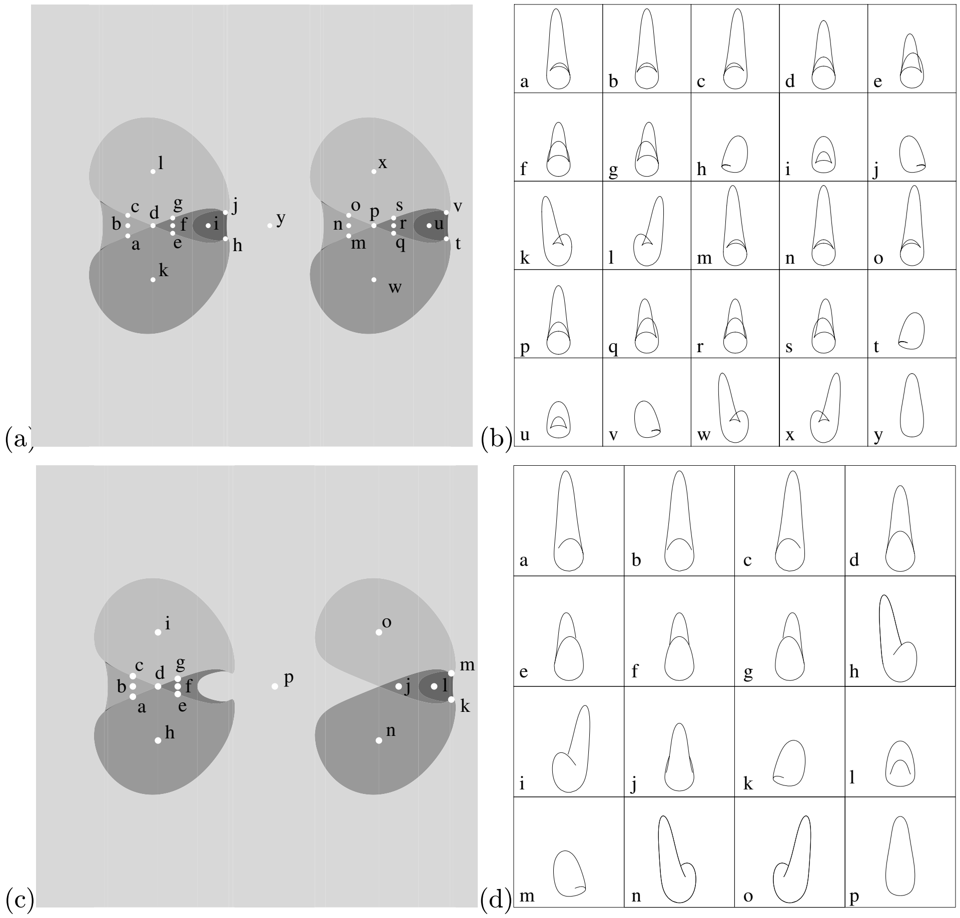

(a)
l
c
d
g
j
b
f
i
y
a
e
h
k
x
o
p
s
v
n
r
u
m
q
t
w
(b)
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
(c)
i
c
d
g
b
f
p
a
e
h
o
m
j
l
k
n
(d)
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p

Figure 22.18. The aspect graph of the bean: (a) the transparent bean; (b) the corresponding aspects; (b) the opaque bean; (d) the corresponding aspects.

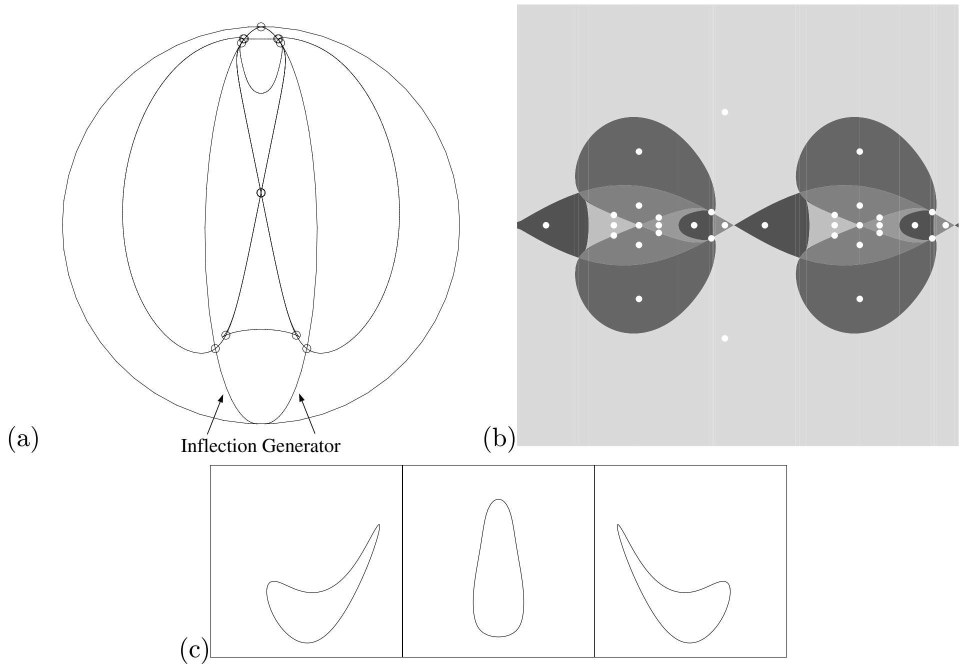


Figure 22.19. The refined aspect graph of the transparent bean: (a) visual event curves; (b) parameter space regions, there are actually 38 regions, but eight of the samples are very close to each other, and only 34 samples can be visually distinguished in this figure; (c) the three aspects corresponding to region y of Figure 22.18.

22.3 Aspect Graphs and Object Recognition

Approximate aspect graphs of polyhedra have been successfully used in recognition tasks [**?**; **?**; **?**].

22.4 Notes

The material in this chapter is (no wonder..) largely based on the work of Koenderink and Van Doorn, including the seminal papers that introduced the idea of an aspect graph (albeit under a different name) [**?**; **?**], the article that presents in a very accessible manner the geometric foundations of this shape representation [**?**] (see also the articles of Platonova [**?**] and Kergosien [**?**]), and of course the somewhat more demanding (but how so rewarding for the patient student) book [**?**]. See also the article by Callahan and Weiss [**?**] for an excellent introduction to aspect graphs. Genericity, singularity and catastrophe theories are discussed in many books, including [**?**; **?**; **?**]. See also [**?**] for a discussion of why chairs wobble and [**?**] for an in-depth discussion of this argument. The algorithm presented in Section 22.2 is due to Petitjean, Ponce and Kriegman [**?**], and the material in this section is largely based on this article. It relies heavily on the global numerical method of *homotopy continuation* [**?**] for finding all roots (including complex roots as well as those at infinity) of a square system of multivariate polynomial equations. Symbolic methods such as multivariate resultants [**?**; **?**; **?**; **?**] and cylindrical algebraic decomposition [**?**; **?**] exist as well, and they have been used by Rieger [**?**; **?**; **?**] in a different algorithm for constructing the aspect graph of an algebraic surface.

We have evaded several issues in this chapter. What happens when the surface that bounds a solid is not generic? Consider for example a manufactured part bounded by planar or quadric surfaces (or by the degree-four algebraic surfaces considered in Section 22.2. But consider too the case of CAD models made of hundreds or thousands of extremely complicated (and yet not generic) bicubic patches. Many algorithms for constructing the aspect graph of a polyhedron under orthographic or perspective projection have been proposed (see, for example, [**?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**; **?**]), and some of them have been implemented. Algorithms for constructing the exact aspect graph of simple curved objects such as solids bounded by quadric surfaces [**?**] and solids of revolution [**?**; **?**; **?**; **?**] have also been proposed and implemented. The algorithms for solids bounded by algebraic surfaces that we presented in this chapter and the approaches based on cylindrical algebraic decomposition [**?**; **?**; **?**] are probably the most general to date, but they are far from being able to tackle CAD models comprised of bicubic patches.

Aspect graphs are unfortunately very large: a polyhedron with n faces has an orthographic aspect graph with $O(n^6)$ cells [**?**]. The size increases to $O(n^6 d^{12})$ for a piecewise-smooth surface made of n polynomial patches of degree d [**?**]. The situation is of course even worse in the perspective case. Should we blame the huge size of aspect graphs on the representation itself? Or is it an artefact of the (combinatorial and/or algebraic) complexity of the underlying surface model, e.g., the number of polyhedral faces or the degree of the patches used to approximate relatively simple free-form surfaces? Noble, Wilson and Ponce [**?**] address the problem of constructing the aspect graph of a solid defined by the zero set of a volumetric density function, e.g., the boundary of an organ in a CT image, and present preliminary results. Another part of the size problem lies in modeling appropriately the optical blur introduced by real lenses together with the finite spatial resolution of cameras. Preliminary efforts at tackling this question can be found in [**?**; **?**; **?**], partly based on recent results about the effect of one-parameter families of deformations on generic surfaces [**?**; **?**].

22.5 Assignments

Exercises

1. Draw the transparent aspect graph of the Flatland object below and the corresponding aspects.
2. Draw the opaque aspect graph of the Flatland object below and the corre-

sponding aspects.

3. Sketch the Gauss map of the object below.
4. Is it possible for an object with a single parabolic curve (such as a banana) to have no cusp of Gauss at all? Why (or why not)?
5. Use an equation-counting argument to justify the fact that contact of order six or greater between lines and surfaces does not occur for generic surfaces. (Hint: count the parameters that define contact.)
6. We saw that the asymptotic curve and its spherical image have perpendicular tangents. *Lines of curvature* are the integral curves of the field of principal directions. Show that these curves and their Gaussian image have parallel tangents.
7. Use the fact that the Gaussian image of a parabolic curve is the envelope of the asymptotic curves intersecting it to give an alternate proof that a pair of cusps is created (or destroyed) in a lip or beak-to-beak event.
8. Aspect graphs of polyhedra, including EEE and EV events.
9. Use an equation counting argument to justify the fact that contact of order 6 or greater between lines and surfaces does not occur for generic surfaces. (Hint: count the parameters that define contact.)

Programming Assignments

1. Write a program to explore multilocal visual events: consider two spheres with different radii and assume orthographic projection. The program should allow you to change viewpoint interactively as well as explore the tangent crossings associated with the limiting bitangent developable.
2. Write a similar program to explore cusp points and their projections. You will have to trace a plane curve.

Part VII

APPLICATIONS AND TOPICS

Chapter 23

RANGE DATA

This chapter discusses *range images* that store, instead of brightness or color information, the depth at which the ray associated with each pixel first intersects the scene observed by a camera. In a sense, a range image is exactly the desired output of stereo, motion, or other shape-from vision modules. In this chapter, however, we will focus our attention on range images acquired by *active* sensors, that project some sort of light pattern on the scene, using it to avoid the difficult and costly problem of establishing correspondences and construct dense and accurate depth pictures. After a brief review of range sensing technology, this chapter will discuss image segmentation, multiple-image registration and three-dimensional model construction, and object recognition, and explore the aspects of these problems that are specific to the range data domain.

23.1 Active Range Sensors

Triangulation-based range finders date back to the early seventies (e.g., [**?**; **?**]). They function along the same principles as passive stereo vision systems, one of the cameras being replaced by a source of controlled illumination (*structured light*) that avoids the correspondence problem mentioned in Chapter 12. For example, a laser and a pair of rotating mirrors may be used to sequentially scan a surface. In this case, as in conventional stereo, the position of the bright spot where the laser beam strikes the surface of interest is found as the intersection of the beam with the projection ray joining the spot to its image. Contrary to the stereo case, however, the laser spot can normally be identified without difficulty since it is in general much brighter than the other scene points (in particular when a filter tuned to the laser wavelength is inserted in front of the camera), altogether avoiding the correspondence problem.

Alternatively, the laser beam can be transformed by a cylindrical lens into a plane of light (Figure 23.1). This simplifies the mechanical design of the range finder since it only requires one rotating mirror. More importantly, perhaps, it shortens the time required to acquire a range image since a laser stripe –the equivalent of a whole image column– can be acquired at each frame. It should be noted that this setup does not introduce matching ambiguities since the spot associated with each pixel can be retrieved as the (unique) intersection of the corresponding projection ray with the plane of light.

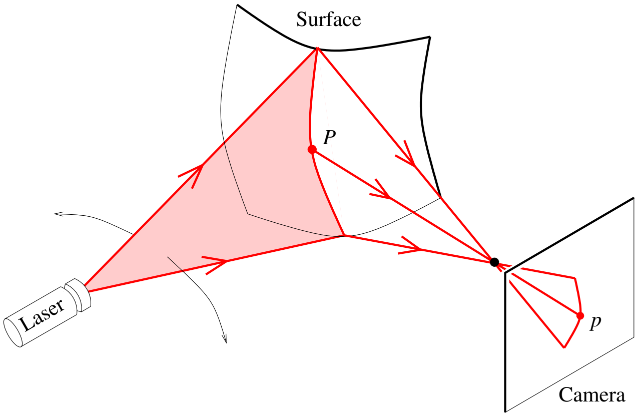


Figure 23.1. A range sensor using a plane of light to scan the surface of an object.

Variants of these two techniques include using multiple cameras to improve measurement accuracy and exploiting (possibly time-coded) two-dimensional light patterns to improve data acquisition speed. The main drawbacks of the active triangulation technology are relatively low acquisition speed, missing data at points where the laser spot is hidden from the camera by the object itself, and missing or erroneous data due to specularities. The latter difficulty is actually common to all active ranging techniques: a purely specular surface will not reflect any light in the direction of the camera unless it happens to lie in the corresponding mirror direction. Worse, the reflected beam may induce secondary reflections giving false depth measurements. Additional difficulties include keeping the laser stripe in focus during the entire scanning procedure, and the loss of accuracy inherent in all triangulation techniques as depth increases.

Several triangulation-based scanners are commercially available today. Figure 23.2 shows an example obtained using the Minolta VIVID range finder, that can acquire a 200×200 range image together with a registered 400×400 color image in 0.6s, within an operating range of 0.6 to 2.5m.

The second main approach to active ranging involves a signal transmitter, a receiver, and electronics for measuring the time of flight of the signal during its round trip from the range sensor to the surface of interest. This is the principle used in the ultrasound domain by the Polaroid range finder, commonly used in autofocus cameras from that brand and in mobile robots, despite the fact that the ultrasound wavelength band is particularly susceptible to false targets due to specular reflections. Time-of-flight laser range finders are normally equipped with a scanning mechanism, and the transmitter and receiver are often coaxial, eliminating the problem of missing data common in triangulation approaches. There are three main classes of time-of-flight laser range sensors: *pulse time delay* tech-

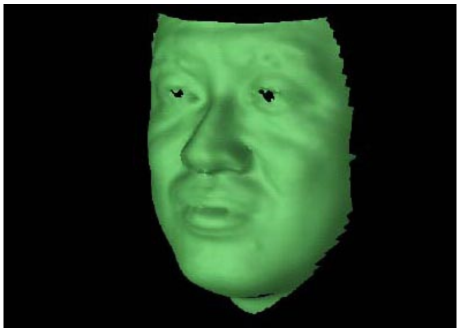


Figure 23.2. Range data captured by the Minolta VIVID scanner. As in most of the figures in this chapter, the mesh of $(x, y, z(x, y))$ points associated with the image is shown in perspective. Reprinted from [**?**], Figure 6.

nology directly measures the time of flight of a laser pulse; *AM phase-shift* range finders measure the phase difference between the beam emitted by an amplitude-modulated laser and the reflected beam, a quantity proportional to the time of flight; finally, *FM beat* sensors measure the frequency shift (or *beat frequency*) between a frequency-modulated laser beam and its reflection, another quantity proportional to the round-trip flight time.

Time-of-flight range finders face the same problems as any other active sensors when imaging specular surfaces. They can be relatively slow due to long integration time at the receiver end. The speed of pulse-time delay sensors is also limited by the minimum resolvable interval between two pulses. AM phase-shift range finders suffer from inherent ambiguities since depth differences corresponding to phase shifts that are multiples of 2π cannot be resolved. This is a relatively minor problem since absolute range can normally be recovered by exploiting spatial coherence in the image, i.e., starting from the image points closest to the sensor, absolute depth can be propagated from one ambiguity interval to the next by a simple region-growing procedure. Compared to triangulation-based systems, time-of-flight sensors have the advantage of offering a greater operating range (up to tens of meters), which is very valuable in outdoor robotic navigation tasks.

Figure 23.3 shows range data acquired by a high-end AM phase-shift scanner [**?**], capable of acquiring 150,000 samples per second at a maximum range of 57m with an accuracy of 34mm.

As noted earlier, many range finders are now available commercially for a wide range of prices and applications. New technologies also continue to emerge, including range sensors equipped with acoustico-optical scanning systems and capable of extremely high image acquisition rates, and range cameras that eliminate scanning altogether, using instead a large array of receivers to analyze a laser pulse covering the entire field of view. Figure 23.4 shows an example of the latter technology, with images acquired by the Zcam range camera from 3DVSystems, which records full-frame registered range and color images at 30Hz with a depth resolution of up

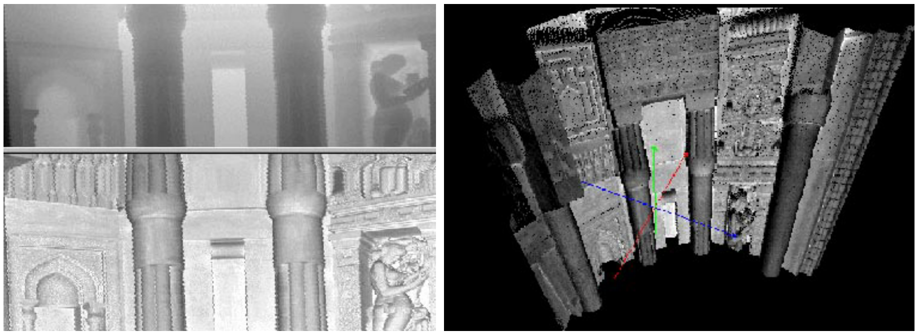


Figure 23.3. Range data captured by the AM phase shift range finder described in [**?**]: (left) range and intensity images; (right) perspective plot of the range data. Reprinted from [**?**], Figure 5.

to 10bits. To learn more, the interested reader should consult [**?**] for an excellent discussion of current range finder technology, including a dozen of representative commercial products.

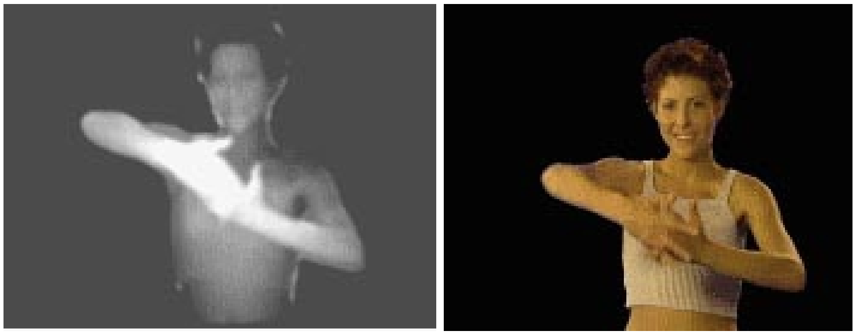


Figure 23.4. Range and color images captured by the Zcam range camera from 3DVSystems. Reprinted from [**?**], Figure 10.

23.2 Range Data Segmentation

This section adapts some of the edge detection and segmentation methods introduced in Chapters **??** and **??** to the specific case of range images. As will be shown in the rest of this section, the fact that surface geometry is readily available greatly simplifies the segmentation process, mainly because this provides objective, physically-meaningful criteria for finding surface discontinuities and merging contiguous patches with a similar shape. But let us start by introducing some elementary notions of *analytical* differential geometry, which will turn out to form the basis for the approach to edge detection in range images discussed in this section.

Technique: Analytical Differential Geometry

Here we revisit the notions of differential geometry introduced in Chapter 4 in an analytical setting. Specifically, we consider a *parametric surface* defined as the smooth (i.e., indefinitely differentiable) mapping $\boldsymbol{x} : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ that associates with any couple (u, v) in the open subset U of $\mathbb{R}^2$ the coordinate vector $\boldsymbol{x}(u, v)$ of a point in some fixed coordinate system. To ensure that the tangent plane is everywhere well defined, we will assume that the partial derivatives $\boldsymbol{x}_u \overset{\text{def}}{=} \partial\boldsymbol{x}/\partial u$ and $\boldsymbol{x}_v \overset{\text{def}}{=} \partial\boldsymbol{x}/\partial v$ are linearly independent. Indeed, let $\boldsymbol{\alpha} : I \subset \mathbb{R} \to U$ denote a smooth planar curve, with $\boldsymbol{\alpha}(t) = (u(t), v(t))$, then $\boldsymbol{\beta} \overset{\text{def}}{=} \boldsymbol{x} \circ \boldsymbol{\alpha}$ is a parameterized space curve lying on the surface. According to the chain rule, a tangent vector to $\boldsymbol{\beta}$ at the point $\boldsymbol{\beta}(t)$ is $u'(t)\boldsymbol{x}_u + v'(t)\boldsymbol{x}_v$, and it follows that the plane tangent to the surface in $\boldsymbol{x}(u, v)$ is parallel to the vector plane spanned by the vectors $\boldsymbol{x}_u$ and $\boldsymbol{x}_v$. The (unit) surface normal is thus

$$\boldsymbol{N} = \frac{1}{|\boldsymbol{x}_u \times \boldsymbol{x}_v|}(\boldsymbol{x}_u \times \boldsymbol{x}_v).$$

Let us consider a vector $\boldsymbol{t} = u'\boldsymbol{x}_u + v'\boldsymbol{x}_v$ in the tangent plane at the point $\boldsymbol{x}$. It is easy to show that the second fundamental form is given by

$$\mathrm{II}(\boldsymbol{t}, \boldsymbol{t}) = \boldsymbol{t} \cdot d\boldsymbol{N}(\boldsymbol{t}) = eu'^2 + 2fu'v' + gv'^2,$$

where[1]

$$e = -\boldsymbol{N} \cdot \boldsymbol{x}_{uu}, f = -\boldsymbol{N} \cdot \boldsymbol{x}_{uv}, g = -\boldsymbol{N} \cdot \boldsymbol{x}_{vv}.$$

Now if we define the *first fundamental form* as the bilinear form that associates with two vectors in the tangent plane their dot product, i.e.,

$$\mathrm{I}(\boldsymbol{u}, \boldsymbol{v}) \overset{\text{def}}{=} \boldsymbol{u} \cdot \boldsymbol{v},$$

then we have

$$\mathrm{I}(\boldsymbol{t}, \boldsymbol{t}) = |\boldsymbol{t}|^2 = Eu'^2 + 2Du'v' + Gv'^2,$$

where

$$E = \boldsymbol{x}_u \cdot \boldsymbol{x}_u, F = \boldsymbol{x}_u \cdot \boldsymbol{x}_v, G = \boldsymbol{x}_v \cdot \boldsymbol{x}_v.$$

It follows immediately that the normal curvature in the direction $\boldsymbol{t}$ is given by

$$\kappa_{\boldsymbol{t}} = \frac{\mathrm{II}(\boldsymbol{t}, \boldsymbol{t})}{\mathrm{I}(\boldsymbol{t}, \boldsymbol{t})} = \frac{eu'^2 + 2fu'v' + gv'^2}{Eu'^2 + 2Du'v' + Gv'^2}.$$

Likewise, it is easily shown that the matrix associated with the differential of the Gauss map *in the basis* $(\boldsymbol{x}_u, \boldsymbol{x}_v)$ of the tangent plane is

$$d\boldsymbol{N}(\boldsymbol{t}) = \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} E & F \\ F & G \end{pmatrix}^{-1};$$

thus, since the Gaussian curvature is equal to the determinant of the operator $d\boldsymbol{N}$, it is given by

$$K = \frac{eg - f^2}{EG - F^2}.$$

---

[1] The definition of e, f and g is in keeping with the orientation conventions defined in Chapter 4. These coefficients are usually defined with opposite signs (e.g. [?; ?]).

Asymptotic and principal directions are also easily found by using this parameterization: since an asymptotic direction verifies $\mathrm{II}(\boldsymbol{t},\boldsymbol{t})=0$, the corresponding values of u' and v' are the (homogeneous) solutions of $eu'^2+2fu'v'+gv'^2=0$. The principal directions, on the other hand, can be shown to verify

$$\begin{vmatrix} v'^2 & -u'v' & u'^2 \\ E & F & G \\ e & f & g \end{vmatrix} = 0. \tag{23.2.1}$$

Example 1. An important example of parametric surface is provided by *Monge patches*: consider the surface $\boldsymbol{x}(u,v)=(u,v,h(u,v))$. In this case we have

$$\begin{cases} \boldsymbol{N} = \dfrac{1}{(1+h_u^2+h_v^2)^{1/2}} \begin{pmatrix} -h_u \\ -h_v \\ 1 \end{pmatrix}, \\ E = 1+h_u^2, F = h_u h_v, G = 1+h_v^2, \\ e = -\dfrac{h_{uu}}{(1+h_u^2+h_v^2)^{1/2}}, f = -\dfrac{h_{uv}}{(1+h_u^2+h_v^2)^{1/2}}, g = -\dfrac{h_{vv}}{(1+h_u^2+h_v^2)^{1/2}}, \end{cases}$$

and the Gaussian curvature has a simple form:

$$K = \frac{h_{uu}h_{vv}-h_{uv}^2}{(1+h_u^2+h_v^2)^2}.$$

Example 2. Another fundamental example is provided by the local parameterization of a surface in the coordinate system formed by its principal directions. This is of course a special case of a Monge patch. Writing that the origin of the coordinate system lies in the tangent plane immediately yields $h(0,0)=h_u(0,0)=h_v(0,0)=0$. As expected, the normal is simply $\boldsymbol{N}=(0,0,1)^T$ at the origin, and the first fundamental form is the identity there.

As shown in the exercises, it follows easily from (23.2.1) that a necessary and sufficient condition for the coordinate curves of a parameterized surface to be principal directions is that $f=F=0$ (this implies, for example, that the lines of curvature of a surface of revolution are its meridians and parallels). In our context we already know that $F=0$ and this condition reduces to $h_{uv}(0,0)=0$. The principal curvatures in this case are simply $\kappa_1=e/E=-h_{uu}(0,0)$ and $\kappa_2=g/G=-h_{vv}(0,0)$.

In particular, we can write a Taylor expansion of the height function in the neighborhood of $(0,0)$ as

$$h(u,v)=h(0,0)+uh_u(0,0)+vh_v(0,0)+\frac{1}{2}(u,v)\begin{pmatrix} h_{uu}(0,0) & h_{uv}(0,0) \\ h_{uv}(0,0) & h_{vv}(0,0) \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix}+\varepsilon(u^2+v^2)^{3/2},$$

which shows that the best second-order approximation to the surface in this neighborhood is the paraboloid defined by

$$h(u,v)=-\frac{1}{2}(\kappa_1 u^2+\kappa_2 v^2),$$

i.e., the expression already encountered in Chapter 4.

23.2.1 Finding Step and Roof Edges in Range Images

This section presents a method for finding instances of various types of edge models in range images [**?**]. This technique combines tools from analytical differential geometry and scale-space image analysis to detect and locate depth and orientation discontinuities in range data. Figure 23.5 shows a 128×128 range image of a motor oil bottle that will serve to illustrate the concepts introduced in this section. This picture was acquired using the INRIA range finder [**?**], with a depth accuracy of about 0.5mm.

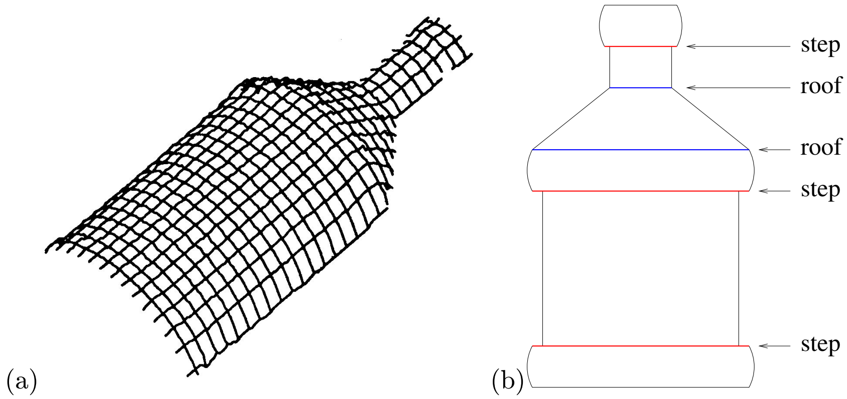


Figure 23.5. An oil bottle: (a) a range image of the bottle and (b) a sketch of its depth and orientation discontinuities.

The surface of the oil bottle presents two types of surface discontinuities: *steps*, where the actual depth is discontinuous, and *roofs*, where the depth is continuous but the orientation changes abruptly. As shown in the next section, it is possible to characterize the behavior of analytical models of step and roof edges under Gaussian smoothing and to show that they respectively give rise to parabolic points and extrema of the dominant principal curvature in the corresponding principal direction. This is the basis for the multi-scale edge detection scheme outlined in Algorithm 23.1 below.

Edge Models

In the neighborhood of a discontinuity, the shape of a surface changes much faster in the direction of the discontinuity than in the orthogonal direction. Accordingly, we will assume in the rest of this section that the direction of the discontinuity is one of the principal directions, with the corresponding (dominant) principal curvature changing rapidly in this direction, while the other one remains roughly equal to zero. This will allow us to limit our attention to *cylindrical* models of surface discontinuities, i.e., models of the form $z(x, y) = h(x)$. These models are of course only intended to be valid in the neighborhood of an edge, with the direction of the

1. Smooth the range image with Gaussian distributions at a set of scales σ_i $(i = 1, \ldots, 4)$. Compute the principal directions and curvatures at each point of the smoothed images $z_{\sigma_i}(x, y)$.
2. Mark in each smoothed image $z_{\sigma_i}(x, y)$ the zero-crossings of the Gaussian curvature and the extrema of the dominant principal curvature in the corresponding principal direction.
3. Use the analytical step and roof models to match the features found across scales and output the points lying on these surface discontinuities.

Algorithm 23.1: *The model-based edge-detection algorithm of Ponce and Brady [?].*

x, z plane being aligned with the corresponding dominant principal direction.

In particular, a step edge can be modeled by two sloped half-planes separated by a vertical gap, with normals in the $x - z$ plane. This model is cylindrical and it is sufficient to study its univariate formulation (Figure 23.6(left)), whose equation is

$$z = \begin{cases} k_1 x + c & \text{when} \quad x < 0, \\ k_2 x + c + h & \text{when} \quad x > 0. \end{cases} \tag{23.2.2}$$

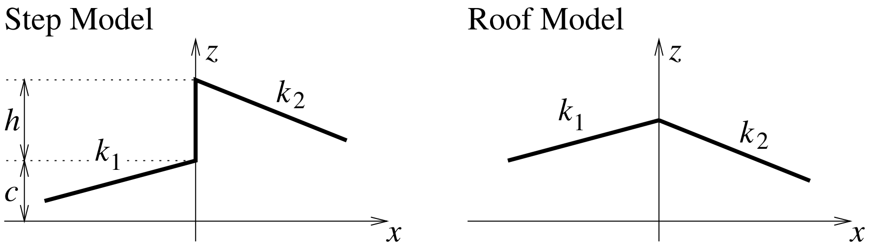


Figure 23.6. Edge models: a step consists of two half-planes separated by a distance h at the origin, and a roof consists of two half-planes meeting at the origin with different slopes. After [?, Figure 4].

In this expression, c and h are constants, h measuring the size of the gap and k_1 and k_2 the slopes of the two half-planes. Introducing the new constants $k = (k_1 + k_2)/2$ and $\delta = k_2 - k_1$, it is easy to show (see exercises) that convolving the z function with the second derivative of a Gaussian yields

$$z''_\sigma \stackrel{\text{def}}{=} \frac{\partial^2}{\partial \sigma^2} G_\sigma * z = \frac{1}{\sigma\sqrt{2\pi}}(\delta - \frac{hx}{\sigma^2}) \exp(-\frac{x^2}{2\sigma^2}). \tag{23.2.3}$$

In particular, the corresponding curvature κ_σ vanishes in $x_\sigma = \sigma^2\delta/h$. This point is only at the origin when $k_1 = k_2$ and its position is a quadratic function of σ otherwise. This suggests identifying step edges with zero-crossings of one of the principal curvatures (or equivalently of the Gaussian curvature), whose position changes with scale. To characterize qualitatively the behavior of these features as a function of σ, let us also note that since $z''_\sigma = 0$ in x_σ, we have

$$\frac{\kappa''_\sigma}{\kappa'_\sigma}(x_\sigma) = \frac{z''''_\sigma}{z''_\sigma}(x_\sigma) = -2\frac{\delta}{\sigma};$$

in other words, the ratio of the second and first derivatives of the curvature is independent of σ.

An analytical model for roof edges is obtained by taking $h = 0$ and $\delta \neq 0$ in the step model (Figure 23.6(right)). In this case, it is easy to show (see exercises) that

$$\kappa_\sigma = \frac{1}{\sigma\sqrt{2\pi}} \frac{\delta \exp(-\dfrac{x^2}{2\sigma^2})}{\left[1 + \left(k + \dfrac{\delta}{\sqrt{2\pi}} \displaystyle\int_0^{x/\sigma} \exp(-\dfrac{u^2}{2})du\right)^2\right]^{3/2}}. \tag{23.2.4}$$

It follows that, when $x_2 = \lambda x_1$ and $\sigma_2 = \lambda\sigma_1$, we must have $\kappa_{\sigma_2}(x_2) = \kappa_{\sigma_1}(x_1)/\lambda$. In turn, the maximum value of $|\kappa_\sigma|$ must be inversely proportional to σ, and it is reached at a point whose distance from the origin is proportional to σ. This maximum tends toward infinity as σ tends toward zero, indicating that roofs can be found as local curvature extrema. In actual range images, these extrema should be sought in the direction of the dominant principal direction, in keeping with our assumptions about local shape changes in the vicinity of surface edges.

Computing the Principal Curvatures and Directions

According to the models derived in the previous section, instances of step and roof edges can be found as zero crossings of the Gaussian curvature and extrema of the dominant principal curvature in the corresponding direction. Computing these differential quantities requires estimating the first and second partial derivatives of the depth function at each point of a range image. This can be done, as in Chapter **??**, by convolving the images with the derivatives of a Gaussian distribution. However, range images are different from usual pictures: for example, the pixel values in a photograph are usually assumed to be piecewise constant in the neighborhood of step edges,[2] which is justified for Lambertian objects since the shape of a surface is, to first order, piecewise-planar near an edge, with a piecewise-planar intensity in that case. On the other hand, piecewise-constant (local) models of range images are of course unsatisfactory. Likewise, the maximum values of contrast along the significant edges of a photograph are usually assumed to have roughly the same magnitude. In range images, however, there are two different types of step edges: the large depth discontinuities that separate solid objects from each other and from their background, and the much smaller gaps that usually separate patches of the same surface.

---

[2] This corresponds to taking $k_1 = k_2 = 0$ in the model given in the previous section; note that in that case zero crossings do not move as scale changes.

The edge detection scheme discussed in this section is aimed at the latter class of discontinuities. Blindly applying Gaussian smoothing across object boundaries will introduce radical shape changes that may overwhelm the surface details we are interested in (Figure 23.7(top and middle)).

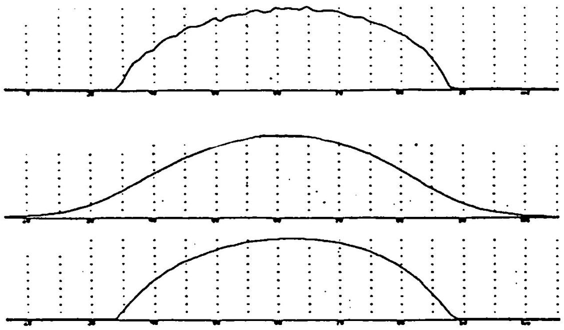


Figure 23.7. Smoothing a range image. Top: a slice of the range image shown in Figure 23.5. The background has been thresholded away. Middle: result of Gaussian smoothing. Bottom: smoothing using computational molecules. Reprinted from [**?**], Figure 14.

This suggests finding the major depth discontinuities first (thresholding will suffice in many cases), then somehow restricting the smoothing process to the surface patches enclosed by these boundaries. This can be achieved by convolving the range image with *computational molecules* [**?**], i.e., linear templates that, added together, form a 3×3 averaging mask, e.g.,

$$\begin{array}{ccc}\boxed{1}&&\\&\boxed{2}&\\&&\boxed{1}\end{array} + \begin{array}{|c|c|c|}\hline 2&4&2\\\hline\end{array} + \begin{array}{|c|}\hline 2\\\hline 4\\\hline 2\\\hline\end{array} + \begin{array}{ccc}&&\boxed{1}\\&\boxed{2}&\\\boxed{1}&&\end{array} = \begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&12&2\\\hline 1&2&1\\\hline\end{array}.$$

Repeatedly convolving the image with the 3×3 mask (normalized so its weights add to one) yields, according to the central limit theorem, a very good approximation of Gaussian smoothing with a mask whose σ value is proportional to $\sqrt{n}$ after n iterations. To avoid smoothing across discontinuities, the molecules crossing these discontinuities are not used, while the remaining ones are once again normalized so the total sum of the weights is equal to one. The effect is shown in Figure 23.7(bottom).

After the surface has been smoothed, the derivatives of the height function can be computed via finite differences. The gradient of the height function is computed by convolving the smoothed image with the masks:

$$\frac{\partial}{\partial x} = \frac{1}{6}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad \text{and} \quad \frac{\partial}{\partial y} = \frac{1}{6}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array},$$

and the Hessian is computed by convolving the smoothed image with the masks

$$\frac{\partial^2}{\partial x^2} = \frac{1}{3}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline 1 & -2 & 1 \\ \hline 1 & -2 & 1 \\ \hline \end{array}, \quad \frac{\partial^2}{\partial x \partial y} = \frac{1}{4}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad \text{and} \quad \frac{\partial^2}{\partial y^2} = \frac{1}{3}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline -2 & -2 & -2 \\ \hline 1 & 1 & 1 \\ \hline \end{array}.$$

Once the derivatives are known, the principal directions and curvatures are easily computed. Figure 23.8 shows the two sets of principal directions found for the oil bottle after 20 iterations of the molecules. As expected, they lie along the meridians and parallels of this surface of revolution.

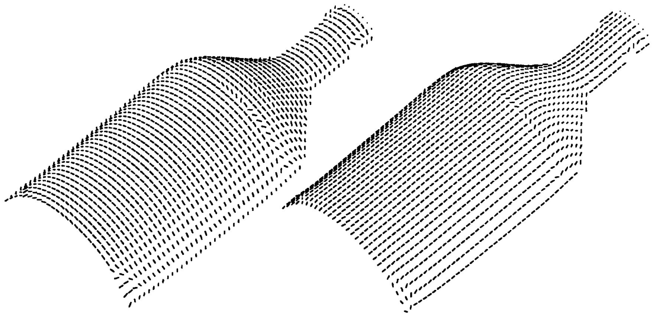


Figure 23.8. The two principal direction fields for the oil bottle. Reprinted from [?], Figure 18.

Matching Features Across Scales

Given the principal curvatures and directions, parabolic points can be detected as (non-directional) zero-crossings of the Gaussian curvature, while local extrema of the dominant curvature along the corresponding principal direction can be found using the non-maximum suppresion techniques discussed in Chapter **??**. Figure 23.9(a) shows the features found after 20, 40, 60, and 80 iterations of the molecule-based smoothing operator. Although there is a considerable amount of noise at fine resolutions (e.g., after 20 iterations only), the situation improves as smoothing proceeds. Features due to noise can also be eliminated, at least in part, via thresholding of the zero-crossing slope for parabolic points, and of the curvature magnitude for extrema of principal curvatures (Figure 23.9(b)).

Nonetheless, experiments show that smoothing and thresholding are not sufficient to eliminate as much as possible all irrelevant features. In particular, as

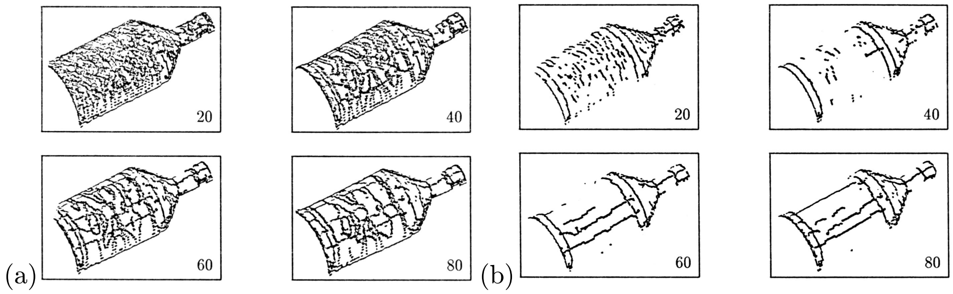


Figure 23.9. Features found at various scales (a) before and (b) after thresholding. Note that the thresholds in (b) have been chosen empirically to eliminate most false features while retaining those corresponding to true surface discontinuities. Still, artefacts such as the extrema of curvature parallel to the axis of the bottle subsist. Reprinted from [?], Figure 12.

illustrated by Figure 23.9, curvature extrema parallel to the axis of the oil bottle show up more and more clearly as smoothing proceeds. These are due to the fact that points near the occluding boundary of the bottle do not get smoothed as much by the computational molecules as points closer to its center, giving rise to artificial curvature extrema.

A multi-scale approach to edge detection solves this problem. Features are tracked from coarse to fine scales, all features at a given scale not having an ancestor at a coarser one being eliminated. The evolution of the principal curvatures and their derivatives is also monitored. Surviving parabolic features such that the ratio $\kappa''_\sigma/\kappa'_\sigma$ remains (roughly) constant across scales are output as step edge points, while directional extrema of the dominant curvature such that $\sigma\kappa_\sigma$ remains (roughly) constant are output as roof points. Finally, since, for both our models, the distance between the true discontinuity and the corresponding zero crossing or extremum increases with scale, the finest scale is used for edge localization. Figure 23.10 shows the results of applying this strategy to the oil bottle and a human face mask.

23.2.2 Segmenting Range Images into Planar Regions

We saw in the last section that edge detection is implemented by quite different processes in photographs and range data. The situation is similar for image segmentation into regions. In particular, meaningful segmentation criteria are elusive in the intensity domain because pixel brightness is only a cue to physical properties such as shape or reflectance. In the range domain however, geometric information is directly available, making it possible to use, say, the average distance between a set of surface points and the plane best fitting them as an effective segmentation criterion. The region growing technique of Faugeras and Hebert [?] is a good example of this approach. This algorithm iteratively merges planar patches by maintaining a graph whose nodes are the patches and arcs associated with their common

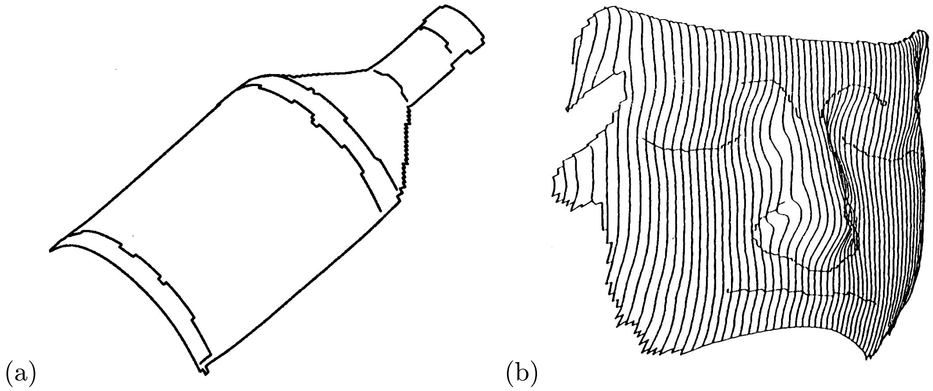


Figure 23.10. Edge detection results: (a) the three step edges and two roof discontinuities of the oil bottle have been correctly identified; (b) the eye, mouth, nose and brow boundaries of a mask have been found as roof edges. Reprinted from [?], Figures 16 and 22.

boundary link adjacent patches. Each arc is assigned a cost corresponding to the average error between the points of the two patches and the plane best fitting these points. The best arc is always selected, and the corresponding patches are merged. Note that the remaining arcs associated with these patches must be deleted while new arcs linking the new patch to its neighbors are introduced. The situation is illustrated by Figure 23.11.

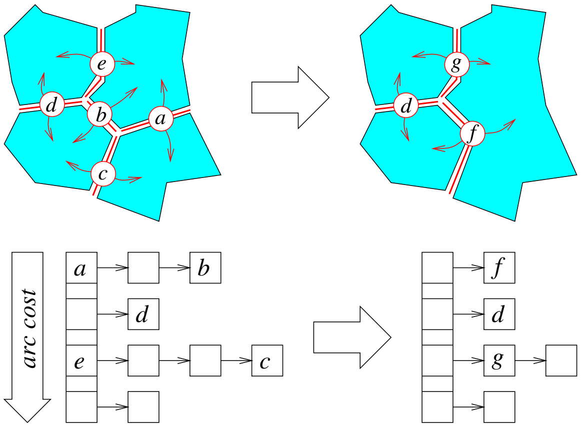


Figure 23.11. This diagram illustrates one iteration of the region growing process during which the two patches incident to the minimum-cost arc labelled a are merged. The heap shown in the bottom part of the figure is updated as well: the arcs a, b, c and e are deleted, and two new arcs f and g are created and inserted in the heap.

The graph structure is initialized by using a triangulation of the range data, and it is efficiently updated by maintaining a heap of active arcs. The triangulation can either be constructed directly from a range image (by splitting the quadrilaterals associated with the pixels along one of their diagonals), or from a global surface model constructed from multiple images as described in the next section. The heap storing the active arcs can be represented, for example, by an array of buckets indexed by increasing costs, which supports fast insertion and deletion (Figure 23.11(bottom)). Figure 23.12 shows an example, where the complex shape of an automobile part is approximated by 60 planar patches.

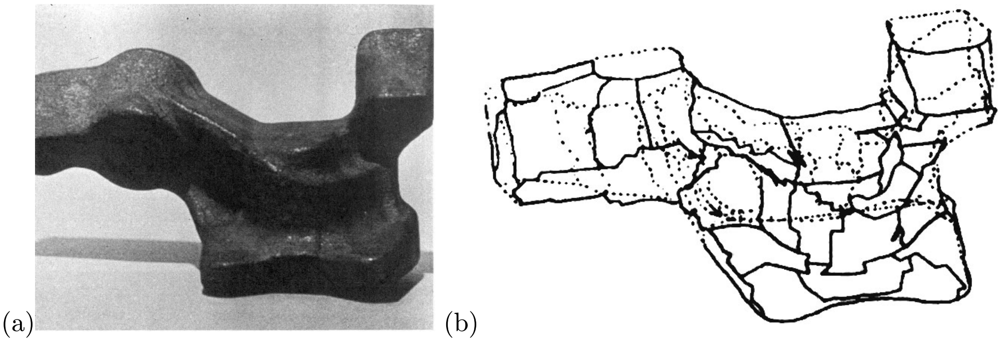

(a) (b)

Figure 23.12. The Renault part: (a) photo of the part and (b) its model. Reprinted from [?], Figures 1 and 6.

23.3 Range Image Registration and Model Construction

Geometric models of real objects are useful in manufacturing, e.g., for process and assembly planning or inspection. Closer to the theme of this book, they are also key components of many object recognition systems, and are more and more in demand in the entertainment industry, as synthetic pictures of real objects now routinely appear in feature films and video games (we will come back to this issue in much greater detail in Chapter 25). Range images are an excellent source of data for constructing accurate geometric models of real objects, but a single picture will, at best, show half of the surface of a given solid, and the construction of complete object models requires the integration of multiple range images. This section addresses the dual problems of registering multiple images in the same coordinate system and fusing the three-dimensional data provided by these pictures into a single integrated surface model.

Before attacking these two problems, let us introduce quaternions, that will provide us with linear methods for estimating rigid transformations from point and plane correspondences in both the registration context of this section and the recognition context of the next one.

Technique: Quaternions

Quaternions were invented by Hamilton [?]. Like complex numbers in the plane, they can be used to represent rotations in space in a very convenient manner. A quaternion q is defined by its *real part*, a scalar a, and its *imaginary part*, a vector $\boldsymbol{\alpha}$ in $\mathbb{R}^3$, and it is usually denoted by $\mathsf{q} = a + \boldsymbol{\alpha}$. This is justified by the fact that real numbers can be identified with quaternions with a zero imaginary part, and vectors can be identified with quaternions with a zero real part, while addition between quaternions is defined by

$$(a + \boldsymbol{\alpha}) + (b + \boldsymbol{\beta}) \stackrel{\text{def}}{=} (a + b) + (\boldsymbol{\alpha} + \boldsymbol{\beta}).$$

The multiplication of a quaternion by a scalar is defined naturally by $\lambda(a + \boldsymbol{\alpha}) \stackrel{\text{def}}{=} \lambda a + \lambda\boldsymbol{\alpha}$, and these two operations give the set of all quaternions the structure of a four-dimensional vector space.

It is also possible to define a multiplication operation that associates with two quaternions the quaternion

$$(a + \boldsymbol{\alpha})(b + \boldsymbol{\beta}) \stackrel{\text{def}}{=} (ab - \boldsymbol{\alpha} \cdot \boldsymbol{\beta}) + (a\boldsymbol{\beta} + b\boldsymbol{\alpha} + \boldsymbol{\alpha} \times \boldsymbol{\beta}).$$

Quaternions, equipped with the operations of addition and multiplication as defined above, form a non-commutative field, whose zero and unit elements are respectively the scalars 0 and 1.

The *conjugate* of the quaternion $\mathsf{q} = a + \boldsymbol{\alpha}$ is the quaternion $\bar{\mathsf{q}} \stackrel{\text{def}}{=} a - \boldsymbol{\alpha}$ with opposite imaginary part. The squared norm of a quaternion is defined by

$$|\mathsf{q}|^2 \stackrel{\text{def}}{=} \mathsf{q}\bar{\mathsf{q}} = \bar{\mathsf{q}}\mathsf{q} = a^2 + |\boldsymbol{\alpha}|^2,$$

and it is easily verified that $|\mathsf{q}\mathsf{q}'| = |\mathsf{q}||\mathsf{q}'|$ for any pair of quaternions q and q'.

Now, it can be shown that the quaternion

$$\mathsf{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\boldsymbol{u}$$

represents the rotation $\mathcal{R}$ of angle θ about the *unit* vector $\boldsymbol{u}$ in the following sense: if $\boldsymbol{\alpha}$ is some vector in $\mathbb{R}^3$, then

$$\mathcal{R}\boldsymbol{\alpha} = \mathsf{q}\boldsymbol{\alpha}\bar{\mathsf{q}}. \tag{23.3.1}$$

Note that $|\mathsf{q}| = 1$ and that $-\mathsf{q}$ also represents the rotation $\mathcal{R}$. Reciprocally, the rotation matrix $\mathcal{R}$ associated with a given unit quaternion $\mathsf{q} = a + \boldsymbol{\alpha}$ with $\boldsymbol{\alpha} = (b, c, d)^T$ is

$$\mathcal{R} = \begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2(bc - ad) & 2(bd + ac) \\ 2(bc + ad) & a^2 - b^2 + c^2 - d^2 & 2(cd - ab) \\ 2(bd - ac) & 2(cd + ab) & a^2 - b^2 - c^2 + d^2 \end{pmatrix},$$

a fact easily deduced from (23.3.1). (Note that the four parameters a, b, c, d are not independent since they satisfy the constraint $a^2 + b^2 + c^2 + d^2 = 1$.)

Finally, if q_1 and q_2 are unit quaternions, and $\mathcal{R}_1$ and $\mathcal{R}_2$ are the corresponding rotation matrices, the quaternions $\mathsf{q}_1\mathsf{q}_2$ and $-\mathsf{q}_1\mathsf{q}_2$ are both representations of the rotation matrix $\mathcal{R}_1\mathcal{R}_2$.

23.3.1 Registering Range Images Using the Iterative Closest-Point Method

Besl and McKay [**?**] have proposed an algorithm capable of registering two sets of three-dimensional points, i.e., of computing the rigid transformation that maps the first point set onto the second one. Their algorithm simply minimizes the average distance between the two point sets by iterating over the following steps: first establish correspondences between scene and model features by matching every scene point to the model point closest to it, then estimate the rigid transformation mapping the scene points onto their matches, and finally apply the computed displacement to the scene. The iterations stop when the change in mean distance between the matched points falls below some preset threshold. Pseudocode for this *iterated closest-point* (or *ICP*) algorithm is given below.

```
Function ICP(Model, Scene);
begin
E' $\leftarrow +\infty$;
(Rot, Trans) $\leftarrow$ Initialize-Registration(Scene, Model);
repeat
   E $\leftarrow$ E';
   Registered-Scene $\leftarrow$ Apply-Registration(Scene, Rot, Trans);
   Pairs $\leftarrow$ Return-Closest-Pairs(Registered-Scene, Model);
   (Rot, Trans, E') $\leftarrow$ Update-Registration(Scene, Model, Pairs, Rot, Trans);
   until $|$E' $-$ E$| < \tau$;
return (Rot, Trans);
end.
```

Algorithm 23.2: *The iterative closest-point algorithm of Best and McKay [**?**]. The auxiliary function Initialize-Registration uses some global registration method, based on moments for example, to compute a rough initial estimate of the rigid transformation mapping the scene onto the model. The function Return-Closest-Pairs returns the indices (i, j) of the points in the registered scene and the model such that point number j is the closest to point number i. The function Update-Registration estimates the rigid transformation between selected pairs of points in the scene and the model, and the function Apply-Registration applies a rigid transformation to all the points in the scene.*

It can be shown that Algorithm 23.2 always converges to a local minimum of the error E (this is intuitively clear since the registration stage decreases the average error at each iteration, while the closest point determination decreases the individual error as well). This does not guarantee of course, convergence to a global minimum, and a reasonable guess for the rigid transformation sought by the algorithm must be provided. A variety of methods are available for that purpose, including roughly sampling the set of all possible transformations, and using the moments of both the scene and model point sets to estimate the transformation.

Finding the Closest-Point Pairs

At every iteration of the algorithm, finding the closest point M in the model to a given (registered) scene point S takes (naively) $O(n)$ time, where n is the number of model points. In fact, various algorithms can be used to answer such a nearest-neighbor query in $\mathbb{R}^3$ in $O(\log n)$ time at the cost of additional preprocessing of the model, using for example *k-d trees* [**?**] (for which the logarithmic query time only holds on average) or more complex data structures. For example, the general randomized algorithm of [**?**] takes preprocessing time $O(n^{2+\varepsilon})$, where ε is an arbitrarily small positive number, and query time $O(\log n)$. The efficiency of repeated queries can also be improved by *caching* the results of previous computations. For example, Simon *et al.* [**?**] store at each iteration of the ICP algorithm the k closest model points to each scene point (a typical value for k is 5). Since the incremental update of the rigid transformation is normally small, it is likely that the closest neighbor of a point after an iteration will be among its k closest neighbors from the previous one. It is in fact possible to determine efficiently and conclusively whether the closest point is in the cached set, see [**?**] for details.

Estimating the Rigid Transformation

Under the rigid transformation defined by the rotation matrix $\mathcal{R}$ and the translation vector $\boldsymbol{t}$, a point $\boldsymbol{x}$ maps onto the point $\boldsymbol{x}' = \mathcal{R}\boldsymbol{x}+\boldsymbol{t}$. Thus, given n pairs of matching points $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$, with $i = 1, \ldots, n$, we seek the rotation matrix $\mathcal{R}$ and translation vector $\boldsymbol{t}$ minimizing the error

$$E = \sum_{i=1}^{n} |\boldsymbol{x}'_i - \mathcal{R}\boldsymbol{x}_i - \boldsymbol{t}|^2.$$

Let us first note that the value of $\boldsymbol{t}$ minimizing E must satisfy

$$0 = \frac{\partial E}{\partial \boldsymbol{t}} = -2\sum_{i=1}^{n}(\boldsymbol{x}'_i - \mathcal{R}\boldsymbol{x}_i - \boldsymbol{t}),$$

or

$$\boldsymbol{t} = \boldsymbol{x}'_0 - \mathcal{R}\boldsymbol{x}_0, \quad \text{where} \quad \boldsymbol{x}_0 \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i \quad \text{and} \quad \boldsymbol{x}'_0 \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}'_i \tag{23.3.2}$$

denote respectively the centroids of the two sets of points $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$.

Introducing the centered points $\boldsymbol{y}_i = \boldsymbol{x}_i - \boldsymbol{x}_0$ and $\boldsymbol{y}'_i = \boldsymbol{x}'_i - \boldsymbol{x}_0$ $(i = 1, \ldots, n)$ yields

$$E = \sum_{i=1}^{n} |\boldsymbol{y}'_i - \mathcal{R}\boldsymbol{y}_i|^2,$$

Quaternions can now be used to minimize E as follows: let q denote the quaternion associated with the matrix $\mathcal{R}$, we use the fact that $|\mathsf{q}|^2 = 1$ and the multiplicativity properties of the quaternion norm to write

$$E = \sum_{i=1}^{n} |\boldsymbol{y}'_i - \mathsf{q}\boldsymbol{y}_i\bar{\mathsf{q}}|^2 |\mathsf{q}|^2 = \sum_{i=1}^{n} |\boldsymbol{y}'_i\mathsf{q} - \mathsf{q}\boldsymbol{y}_i|^2.$$

As shown in the exercises, this allows us to rewrite the rotational error as $E = \mathsf{q}^T \mathcal{B} \mathsf{q}$, where $\mathcal{B} = \sum_{i=1}^{n} \mathcal{A}_i^T \mathcal{A}_i$, and

$$\mathcal{A}_i = \begin{pmatrix} 0 & \boldsymbol{y}_i^T - \boldsymbol{y}_i'^T \\ \boldsymbol{y}'_i - \boldsymbol{y}_i & [\boldsymbol{y}_i + \boldsymbol{y}'_i]_\times \end{pmatrix}.$$

Note that the matrix $\mathcal{A}_i$ is antisymmetric with (in general) rank 3, but that the matrix $\mathcal{B}$ will have, in the presence of noise, rank 4. As shown in Chapter 5, minimizing E under the constraint $|\mathsf{q}|^2 = 1$ is a (homogeneous) linear least-squares problem whose solution is the eigenvector of $\mathcal{B}$ associated with the smallest eigenvalue of this matrix. Once $\mathcal{R}$ is known, $\boldsymbol{t}$ is obtained from (23.3.2).

Results

Figure 23.13 shows an example, where two range images of an African mask are matched by the algorithm. The average distance between matches is 0.59mm for this 9cm object.

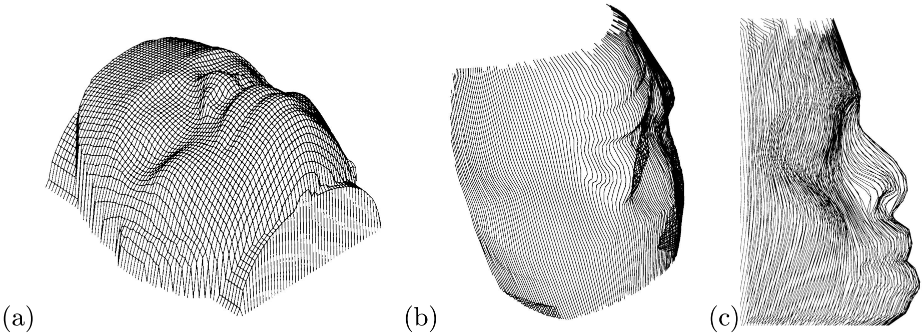


Figure 23.13. Registration results: (a) a range image serving as model for an African mask; (b) a (decimated) view of the model, serving as scene data; (c) a view of the two datasets overlaid after registration. Reprinted from [**?**], Figures 12–14.

This method is not limited to models consisting to clouds of three-dimensional points, but applies as well to any model that supports the construction of closest-point pairs. Figure 23.14 shows an example where a range image is matched to a spline model of the mask. In this case, the patch point closest to a scene point is retrieved by a simple optimization process, initialized at the center of the patch for example (see exercises).

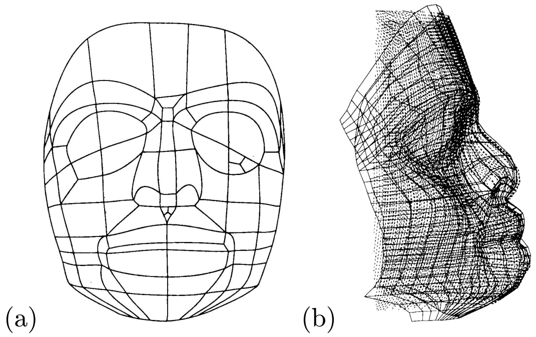


Figure 23.14. More registration results: (a) a parametric surface serving as model for the mask; (b) registration of this model and the range image shown in Figure 23.13. Reprinted from [**?**], Figures 15 and 16.

23.3.2 Fusing Multiple Range Images

Given a set of registered range images of a solid object, it is possible to construct an integrated surface model of this object. In the approach proposed by Curless and Levoy [**?**], this model is constructed as the zero set S of a volumetric density function $D : \mathbb{R}^3 \rightarrow \mathbb{R}$, i.e., as the set of points (x, y, z) such that $D(x, y, z) = 0$. Like any other level set of a continuous density function, S is by construction guaranteed to be a closed, "watertight" surface, although it may have several connected components (Figure 23.15)

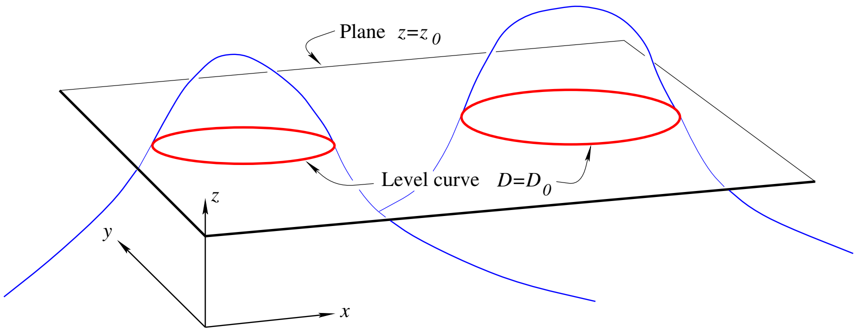


Figure 23.15. A 2D illustration of volumetric density functions and their level sets. In this case, the "volume" is of course the (x, y) plane, and the "surface" is a curve in this plane, with two connected components in the example shown here.

The trick, of course, is to construct an appropriate density function from registered range measurements. Curless and Levoy embed the corresponding surface fragments into a cubic grid, and assign to each cell of this grid, or *voxel*, a weighted sum of the signed distances between its center and the closest point on the surface intersecting it (Figure 23.16(left)). This averaged signed distance is the desired density function, and its zero set can be found using classical techniques, such as the *marching cubes* algorithm develop by Lorensen and Cline [**?**] to extract isodensity surfaces from volumetric medical data.

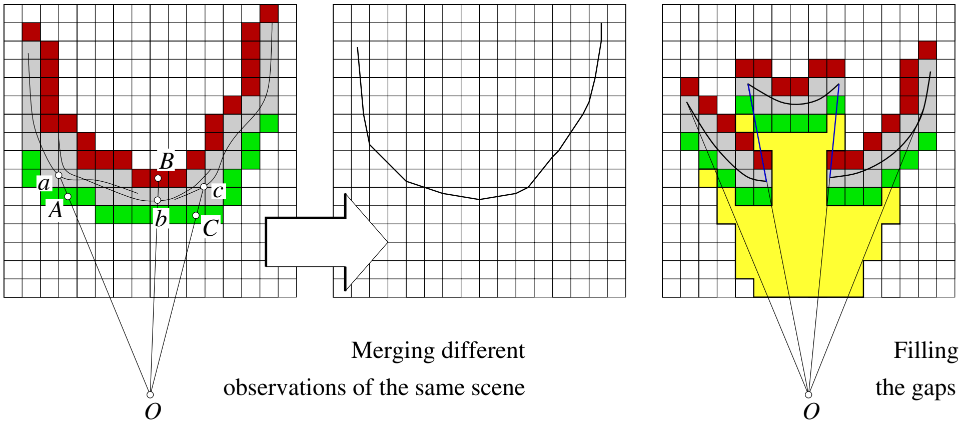


Figure 23.16. A 2D illustration of the Curless-Levoy method for fusing multiple range images. In the left part of the figure, three views observed by the same sensor located at the point O are merged by computing the zero set of a weighted average of the signed distances between voxel centers (e.g., points A, B and C) and surface points (e.g., a, b and c) along viewing rays. In general, distances to different sensors would be used instead. The light grey area in the right part of the figure is the set of voxels marked as empty in the gap-filling part of the procedure.

Missing surface fragments corresponding to unobserved parts of the scene are handled by initially marking all voxels as *unseen*, or equivalently assigning them a depth equal to some large positive value (standing for $+\infty$), then assigning as before to all voxels close to the measured surface patches the corresponding signed distance, and finally carving out (i.e., marking as *empty*, or having a large negative depth standing for $-\infty$) the voxels that lie between the observed surface patches and the sensor (Figure 23.16(right)).

Figure 23.17 shows an example of model built from multpiple range images of a Buddha statuette acquired with a Cyberware 3030 MS optical triangulation scanner, as well as a physical model constructed from the geometric one via stereolithography [**?**].

23.4 Object Recognition

We now turn to actual object recognition from range images. The registration techniques introduced in the previous section will play a crucial role in the two algorithms discussed in this one.

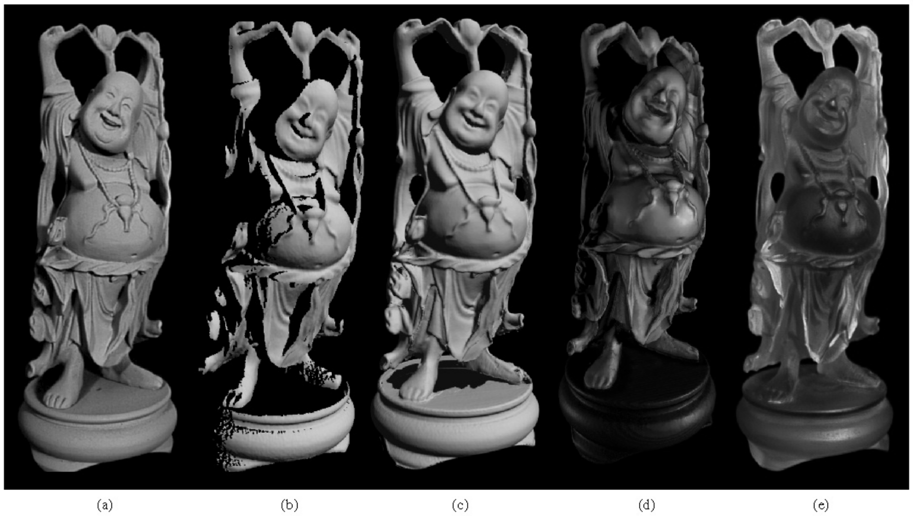


Figure 23.17. 3D Fax of a statuette of a Buddha. From left to right: photograph of the statuette; range image; integrated 3D model; model after hole filling; physical model obtained via stereolithography. Reprinted from [**?**], Figure 10.

23.4.1 Matching Piecewise-Planar Surfaces Using Interpretation Trees

The recognition algorithm proposed by Faugeras and Hebert [**?**] is a recursive procedure exploiting rigidity constraints to efficiently search an interpretation tree for the path(s) corresponding to the best sequence(s) of matches. The basic procedure is given in pseudocode in Algorithm 23.3 below. To correctly handle occlusions (and the fact that, as noted earlier, a range finder will "see", at best, one half of the object facing it), the algorithm must consider, at every stage of the search, the possibility that a model plane may not match any scene plane. This is done by always incorporating in the list of potential matches of a given plane a token "null" plane.

Selecting Potential Matches

The selection of potential matches for a given model plane is based on various criteria depending on the number of correspondences already established, with each new correspondence providing new geometric constraints and more stringent criteria. At the beginning of the search, we only know that a model plane with area A should only be matched to scene planes with a compatible area, i.e., in the range $[\alpha A, \beta A]$. Reasonable values for the two thresholds might be 0.5 and 1.1, which allows for some discrepency between the unoccluded areas, and also affords a degree of occlusion

```
Function Match(model, scene, pairs, rot, trans);
begin
bestpairs ← nil; bestscore ← 0;
for Π in model do
   for Π′ in Potential-Matches(scene, pairs, Π, rot, trans) do
     rot ← Update-Registration-2(pairs, Π, Π′, rot, trans);
     (score, newpairs) ← Match(model−Π, scene−Π′, pairs+(Π,Π′), rot, trans);
     if score>bestscore then bestscore ← score; bestpairs ← newpairs endif;
     endfor;
   endfor;
return bestpairs;
end.
```

Algorithm 23.3: *The plane-matching algorithm of Faugeras and Hebert [?]. The recursive function Match returns the best set of matching plane pairs found by recursively visiting the interpretation tree. It is initially called with an empty list of pairs and nil values for the rotation and translation arguments rot and trans. The auxiliary function Potential-Matches returns the subset of the planes in the scene that are compatible with the model plane* Π *and the current estimate of the rigid transformation mapping the model planes onto their scene matches (see text for details). The auxiliary function Update-Registration-2 uses the matched plane pairs to update the current estimate of the rigid transformation.*

up to 50%.

After the first correspondence has been established, it is still too early to try and estimate the rigid transformation mapping the model onto the scene, but it is clear that the angle between the normals to any matching planes should be (roughly) equal to the angle θ between the normals to the first pair of planes, say lie in the interval $[\theta-\varepsilon, \theta+\varepsilon]$. The normals to the corresponding planes lie in a band of the Gauss sphere, and they can be efficiently retrieved by discretizing this sphere and associating to each cell a bucket that stores the scene planes whose normal falls into it (Figure 23.18).

A second pairing is sufficient to complete determine the rotation separating the model from its instance in the scene: this is geometrically clear (and will be confirmed analytically in the next section) since a pair of matching vectors constrains the rotation axis to lie in the plane bisecting these vectors. Two pairs of matching planes determine the axis of rotation as the intersection of the corresponding bisecting planes, and the rotation angle is readily computed from either of the matches. Given the rotation and a third model plane, one can predict the orientation of the normal to its possible matches in the scene, which can be efficiently recovered using once again the discrete Gauss sphere mentioned before.

After three pairings have been found, the translation can also be estimated and used to predict the distance between the origin and any scene plane matching a

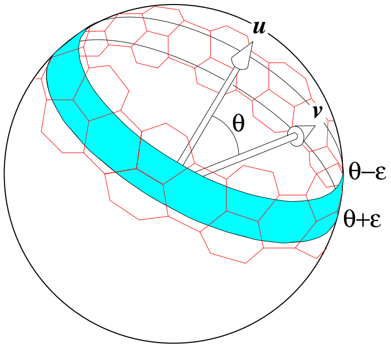


Figure 23.18. Finding all vectors $\boldsymbol{v}$ that make an angle in the $[\theta-\varepsilon, \theta+\varepsilon]$ range with a given vector $\boldsymbol{u}$. It should be noted that the unit sphere does not admit tesselations with an arbitrary level of detail by regular (spherical) polygons. The tesselation shown in the diagram is made of hexagons with unequal edge lengths. See, for example, [**?**, Chap. 16] for a discussion of this problem and various tesselation schemes.

fourth scene plane. The same is true for any further pairing.

Estimating the Rigid Transformation

Let us consider a plane Π defined by the equation $\boldsymbol{n}\cdot\boldsymbol{x}-d=0$ in some fixed coordinate system. Here $\boldsymbol{n}$ denotes the unit normal to the plane and d its (signed) distance from the origin. Under the rigid transformation defined by the rotation matrix $\mathcal{R}$ and the translation vector $\boldsymbol{t}$, a point $\boldsymbol{x}$ maps onto the point $\boldsymbol{x}'=\mathcal{R}\boldsymbol{x}+\boldsymbol{t}$, and Π maps onto the plane Π' whose equation is $\boldsymbol{n}'\cdot\boldsymbol{x}'-d'=0$, with

$$\begin{cases} \boldsymbol{n}'=\mathcal{R}\boldsymbol{n}, \\ d'=\boldsymbol{n}'\cdot\boldsymbol{t}+d. \end{cases}$$

Thus, estimating the rigid transformation that maps n planes Π_i onto the matching planes Π'_i $(i=1,\dots,n)$ amounts to finding the rotation matrix $\mathcal{R}$ that minimizes the error

$$E_r=\sum_{i=1}^{n}|\boldsymbol{n}'_i-\mathcal{R}\boldsymbol{n}_i|^2$$

and the translation vector $\boldsymbol{t}$ that minimizes

$$E_t=\sum_{i=1}^{n}(d'_i-d_i-\boldsymbol{n}'_i\cdot\boldsymbol{t})^2.$$

The rotation $\mathcal{R}$ minimizing E_r can be computed, exactly as in Section 23.4.1, by using the quaternion representation of matrices and solving an eigenvector problem.

The translation vector $\boldsymbol{t}$ minimizing E_t is the solution of a (non-homogeneous) linear least-squares problem, whose solution can be found using the techniques presented in Chapter 5.

Results

Figure 23.19 shows recognition results obtained using a bin of Renault parts such as the one shown in Figure 23.12. The range image of the bin has been segmented into planar patches using the technique presented in Section 23.2.2. The matching algorithm is run three times on the scene, with patches matched during each run removed from the scene before the next iteration. As shown by the figure, the three instances of the part present in the bin are correctly identified, and the accuracy of the pose estimation process is attested by the reprojection into the range image of the model in the computed pose.

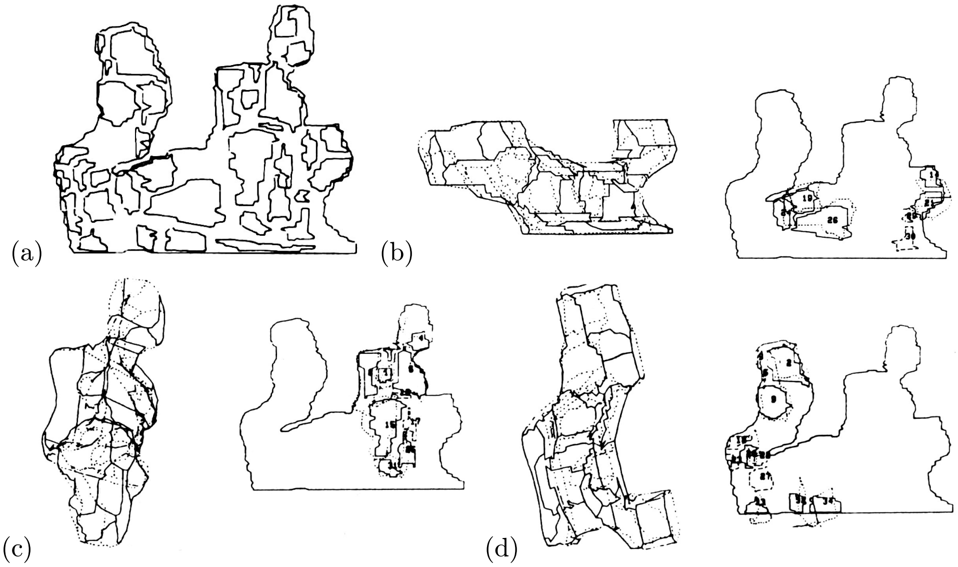


Figure 23.19. Recognition results: (a) a bin of parts, and (b)-(d) the three instances of the Renault part found in that bin. In each case, the model is shown both by itself in the position and orientation estimated by the algorithm, as well as superimposed (dotted lines) in this pose over the corresponding planes of the range image. Reprinted from [**?**], Figures 14–16.

23.4.2 Matching Free-Form Surfaces Using Spin Images

As demonstrated in Section 23.2.1, differential geometry provides a powerful language for describing the shape of a surface *locally*, i.e., in a small neighborhood of each one of its points. On the other hand, the region-growing algorithm discussed in Section 23.2.2 is aimed at constructing a *globally* consistent surface description in terms of planar patches. We introduce in this section a *semi-local* surface representation, the spin image of Johnson and Hebert [**?**; **?**], that captures the shape of a surface in a relatively large neighborhood of each one of its points. As will be shown in the rest of this section, the spin image is invariant under rigid transformations, and it affords an efficient algorithm for pointwise surface matching, thus completely bypassing segmentation in the recognition process.

Spin Image Definition

Let us assume as in Section 23.2.2 that the surface of interest is given in the form of a triangular mesh. The (outward-pointing) surface normal at each vertex can be estimated by fitting a plane to this vertex and its neighbors, turning the triangulation into a net of *oriented points*. Given an oriented point P, the *spin coordinates* of any other point Q can now be defined as the (nonnegative) distance α separating Q from the (oriented) normal line in P and the (signed) distance β from the tangent plane to Q (Figure 23.20).

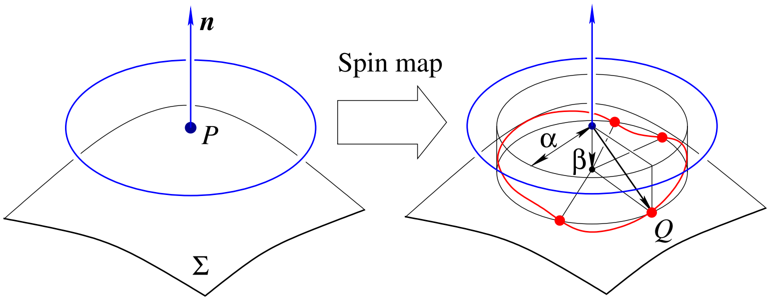


Figure 23.20. Definition of the spin map associated with a surface point P: the spin coordinates (α, β) of the point Q are respectively defined by the lengths of the projections of $\overrightarrow{PQ}$ onto the tangent plane and its surface normal. Note that there are three other points with the same (α, β) coordinates as Q in this example.

Accordingly, the *spin map* $s_P : \Sigma \to \mathbb{R}^2$ associated with P is defined for any point Q on Σ as

$$s_P(Q) \overset{\text{def}}{=} (\underbrace{|\overrightarrow{PQ} \times \boldsymbol{n}|}_{\alpha}, \underbrace{\overrightarrow{PQ} \cdot \boldsymbol{n}}_{\beta}).$$

As shown by Figure 23.20, this mapping is not injective. This is not surprising since the spin map only provides a partial specification of a cylindrical coordinate system: the third coordinate that would normally record the angle between some reference vector in the tangent plane and the projection of $\overrightarrow{PQ}$ into this plane is missing. The principal directions are obvious choices for such a reference vector, but focussing on the spin coordinates avoids their computation, a process that is susceptible to noise since it involves second derivatives and may be ambiguous for (almost) planar or spherical patches.

The *spin image* associated with an oriented point is a histogram of the α, β coordinates in a neighborhood of this point (Figure 23.20(b)). Concretely, the α, β plane is divided into a rectangular array of $\delta\alpha \times \delta\beta$ bins that accumulate the total surface area spanned by points with α, β values in that range.[3] As shown in [**?**] and the exercises, each triangle in the surface mesh maps onto a region of the α, β plane whose boundaries are hyperbola arcs. Its contribution to the spin image can thus be computed by scan-converting this region and assigning to each bin that it traverses the area of the patch where the triangle intersects the annular region of $\mathbb{R}^3$ associated with the bin (Figure 23.21).

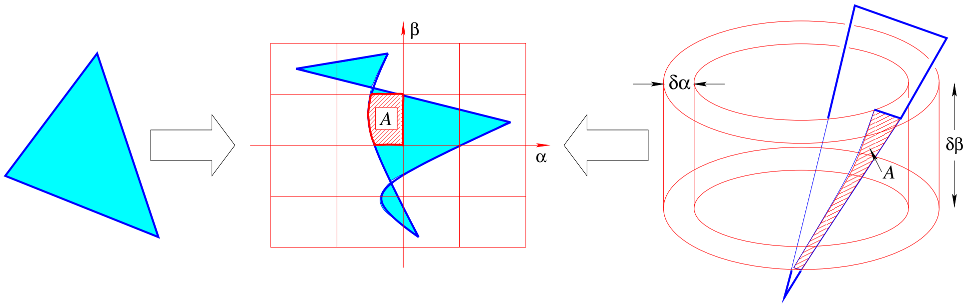


Figure 23.21. Spin image construction: the triangle shown in the left of the diagram maps onto a region with hyperbolic boundaries in the spin image; the value of each bin intersected by this region is incremented by the area of the portion of the triangle that intersects the annulus associated with the bin. After [**?**, Figure 3].

A key parameter of spin images is the support distance d that limits to a sphere of radius d centered in P the range of the *support points* used to construct the image. This sphere must be large enough to provide good descriptive power but small enough to support recognition in the presence of clutter and occlusion. In practice, an appropriate choice for d might be a tenth of the object's diameter [**?**]: thus, as noted earlier, the spin image is indeed a semi-local description of the shape of a surface in an *extended* neighborhood of one of its points.

Robustness to clutter can be improved by limiting the range of surface normals at the support points to a cone of half-angle θ centered in $\boldsymbol{n}$. As in the support distance case, choosing the right value for θ involves a tradeoff between descriptive power and insensitivity to clutter; a value of 60° has empirically been shown to be satisfactory in [**?**]. The last parameter defining a spin image is its size (in pixels), or equivalently, given the support distance, its bin size (in meters). As shown in [**?**], an appropriate choice for the bin size is the average distance between mesh vertices in the model. Figure 23.22 shows the spin images associated with three oriented points on the surface of a rubber duck.

---

[3] The corresponding point sets may actually be divided into several connected components: for example, for small enough values of $\delta\alpha$ and $\delta\beta$ there are four connected components in the example shown in Figure 23.20, corresponding to small patches centered at the points having the same α, β coordinates as Q.

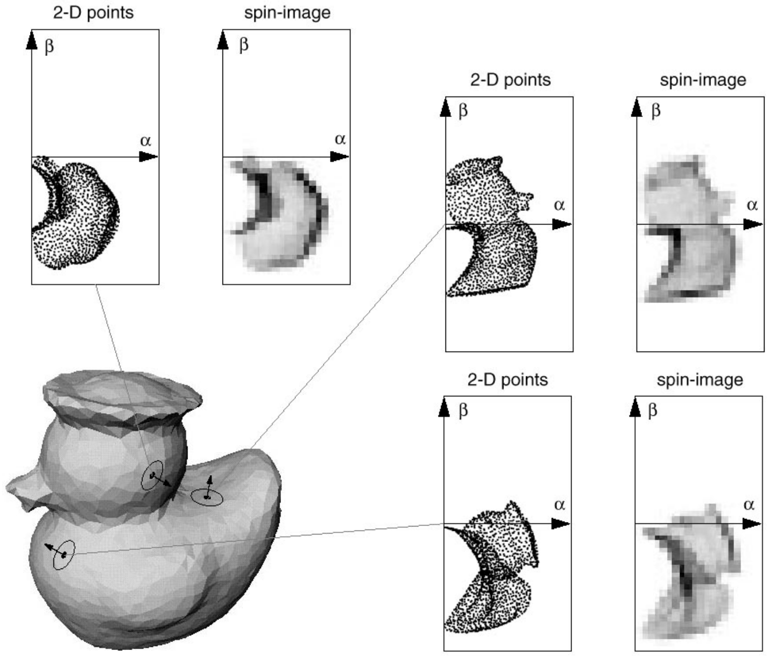


Figure 23.22. Three oriented points on the surface of a rubber duck and the corresponding spin images. The α, β coordinates of the mesh vertices are shown besides the actual spin images. Reprinted from [**?**], Figure 3.

Matching Spin Images

One of the most important features of spin images is that they are (obviously) invariant under rigid transformations. Thus an image comparison technique such as correlation can in principle be used to match the spin images associated with oriented points in the scene and the object model. Things are not that simple, however: we already noted that the spin map is not injective; in general, it is not surjective either, and empty bins (or equivalently zero-valued pixels) may occur for values of α and β that do not correspond to physical surface points (see the blank areas in Figure 23.22 for example). Occlusion may cause the appearance of zero pixels in the scene image, while clutter may introduce irrelevant non-empty bins. It is therefore reasonable to restrict the comparison of two spin images to their common non-zero pixels. In this context, Johnson and Hebert [**?**] have shown that

$$S(\boldsymbol{I}, \boldsymbol{J}) \stackrel{\text{def}}{=} [\text{Arctanh}(C(\boldsymbol{I}, \boldsymbol{J}))]^2 - \frac{3}{N-3}$$

is an appropriate similarity measure for two spin images whose overlap regions contain N pixels and are represented by the vectors $\boldsymbol{I}$ and $\boldsymbol{J}$ of $\mathbb{R}^N$. In this formula,

$C(\boldsymbol{I}, \boldsymbol{J})$ denotes the normalized correlation of the vectors $\boldsymbol{I}$ and $\boldsymbol{J}$, and Arctanh denotes the hyperbolic arc tangent function. Armed with this similarity measure, we can now outline a recognition algorithm that uses spin images to establish pointwise correspondences.

Off-line:

Compute the spin images associated with the oriented points of a surface model and store them into a table.

On-line:

1. Form correspondences between a set of spin images randomly selected in the scene and their best matches in the model table using the similarity measure S to rank-order the matches.
2. Filter and group correspondences using geometric consistency constraints, and compute the rigid transformations best aligning the matched scene and model features.
3. Verify the matches using the ICP algorithm.

Algorithm 23.4: *The algorithm of Johnson and Hebert [**?**; **?**] for pointwise matching of free-form surfaces using spin images.*

The various stages of this algorithm are mostly straightforward. Let us note however that the filtering/grouping step relies on comparing the spin coordinates of model points relative to the other mesh vertices in their group with the spin coordinates of the corresponding scene points relative to their own group. Once consistent groups have been identified, an initial estimate of the rigid transformation aligning the scene and the model is computed from (oriented) point matches using the quaternion-based registration technique described in Section 23.3.1. Finally, consistent sets of correspondences are verified by iteratively spreading the matching process to their neighbors, updating along the way the rigid transformation that aligns the scene and the model.

Results

The matching algorithm presented in the previous section has been extensively tested in recognition tasks with cluttered indoor scenes that contain both industrial parts and various toys [**?**; **?**]. It has also been used in outdoor navigation/mapping tasks with very large datasets covering thousands of squared meters of terrain [**?**]. Figure 23.23 shows sample recognition results in the toy domain.

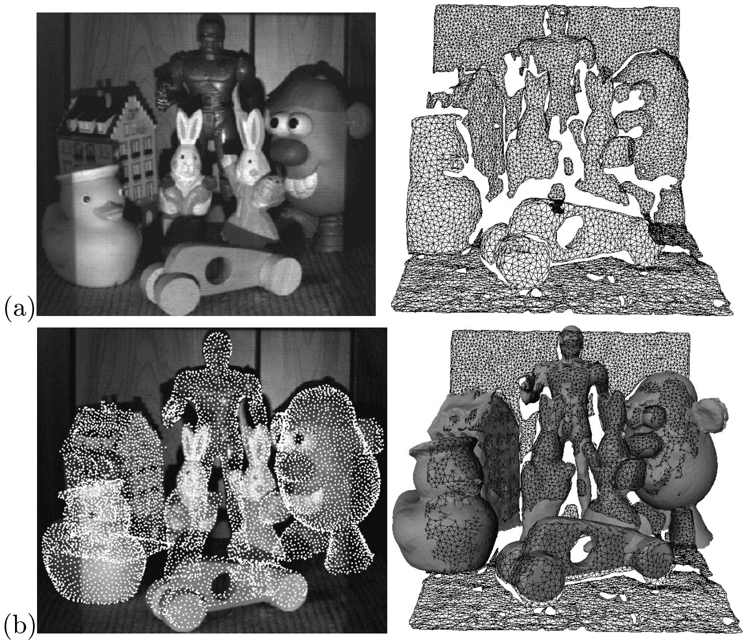


Figure 23.23. Spin-image recognition results: (a) a cluttered image of toys and the mesh constructed from the corresponding range image; (b) recognized objects overlaid on the original pictures.

23.5 Notes

Excellent surveys of active range finding techniques can be found in [?; ?; ?; ?]. The model-based approach to edge detection presented in Section 23.2.1 is only one of the many techniques that have been proposed for segmenting range pictures using notions from differential geometry (see, for example, [?; ?]). An alternative to the computational molecules used to smooth a range image in that section is provided by anisotropic diffusion, where the amount of smoothing at each point depends on the value of the gradient [?].

The method for segmenting surfaces into (almost) planar patches presented in Section 23.2.2 is easily extended to quadric patches (see [?] and the exercises). Extensions to higher-order surface primitives is more problematic, in part because surface fitting is more difficult in that case. There is a vast amount of literature on the latter problem, using superquadrics (e.g., [?; ?; ?]) and algebraic surfaces (e.g, [?; ?; ?]) for example.

Alternatives to the Curless and Levoy [?] approach to the fusion of multiple range images include the Delaunay triangulation algorithm of Boissonnat [?], the zippered polygonal meshes of Turk and Levoy [?] and the crust technique of Amenta *et al.* [?]. The quaternion-based approach to the estimation of rigid transformations described in this chapter was developed independently by Faugeras and Hebert [?] and Horn [?]. The recognition technique discussed in Section 23.4.1 is closely related to other algorithms using interpretation trees to control the combinatorial cost of feature matching in the two- and three-dimensional cases [?; ?; ?; ?].

The spin images discussed in Section 23.4.2 have been used to establish pointwise correspondences between range images and surface models. Related approaches to this problem include the structural indexing method of Stein and Medioni [?] and the point signatures proposed by Chua and Jarvis [?]. A (local) variant of the same idea will be discussed in Chapter 20 in the context of object recognition from photographs [?]. To conclude, let us note that the original algorithm described in Section 23.4.2 has been extended in various directions: a scene can now be matched simultaneously to several models using principal component analysis (see Chapter 20 and [?]), while learning techniques are used to prune false matches in cluttered scenes [?].

23.6 Assignments

Exercises

1. Step model: compute $z_\sigma(x) = G_\sigma * z(x)$, where $z(x)$ is given by (23.2.2). Show that z''_σ is given by (23.2.3). Conclude that $\kappa''_\sigma/\kappa'_\sigma = -2\delta/h$ in the point x_σ where z''_σ and κ_σ vanish.

2. Roof model: show that κ_σ is given by (23.2.4).

3. Use (23.2.1) to show that a necessary and sufficient condition for the coordinate curves of a parameterized surface to be principal directions is that $f = F = 0$.

4. Show that the lines of curvature of a surface of revolution are its meridians and parallels.

5. Calculate the Gaussian curvature of an SHGC.

6. Show that the matrix $\mathcal{A}_i$ constructed in Section 23.3.1 is equal to

$$\mathcal{A}_i = \begin{pmatrix} 0 & \boldsymbol{y}_i^T - \boldsymbol{y}_i'^T \\ \boldsymbol{y}_i' - \boldsymbol{y}_i & [\boldsymbol{y}_i + \boldsymbol{y}_i']_\times \end{pmatrix}.$$

7. As mentioned earlier, the ICP method can be extended to various types of geometric models. We consider here the case of polyhedral models and piecewise parametric patches.
   (a) Give a method for computing the point Q in a polygon that is closest to some point P.
   (b) Give a method for computing the point Q in the parametric patch $\boldsymbol{x} : I \times J \to \mathbb{R}^3$ that is closest to some point P. Hint: use Newton iterations.

8. Develop a linear least-squares method for fitting a quadric surface to a set of points under the constraint that the quadratic form has unit Frobenius form.

9. Show that a surface triangle maps onto a patch with hyperbolic edges in α, β space.

Programming Assignments

The datasets for the following machine problems can be found on the CD companion of this book.

1. Implement molecule-based smoothing and the computation of principal directions and curvatures.

2. Implement the region-growing approach to plane segmentation described in this chapter.

3. Implement an algorithm for computing the lines of curvature of a surface from its range image. Hint: use a curve-growing algorithm analogous to the region-growing algorithm for plane segmentation.

4. Implement the Besl-McKay ICP registration algorithm.

5. Marching squares in the plane: develop and implement an algorithm for finding the zero set of a planar density function. Hint: work out the possible ways a curve may intersect the edges of a pixel, and use linear interpolation along these edges to identify the zero set.

6. Implement the registration part of the Faugeras-Hebert algorithm.

Chapter 24

APPLICATION: FINDING IN DIGITAL LIBRARIES

Many collections of pictures are being digitized in the hope of better conservation, easier distribution, and better access. Once the pictures have been digitized, there remains the problem of finding the image we want in the collection. This is a subtle problem, because preparing a good text description of an image is difficult, so that text indexing systems are not always very much help. Furthermore, some collections of digital images are quite disorganised — for example, there is little prospect of preparing a text index for each image on the web, which contains a huge number of pictures.

Finding something in a collection of pictures or of video clips is a problem that appears in a variety of contexts. The increasing use of digital media is generating huge searchable collections which need to be managed. This **digital asset management** is a generalisation of the search problem. For example, digital projectors mean that film distributors can: easily change the film on display at a particular cinema; add or remove scenes according to local sensibilities; show alternate endings; etc. Presumably distributors will do this in ways that maximise revenue, but to do so they need to make sure that the particular item of digital content is where it needs to be, when it is wanted.

This is an interesting problem because collections are often enormous. Some of the collections of images described in [**?**] contain tens of millions of pictures. Indexing a large collection by hand involves a substantial volume of work. Furthermore, there is the prospect of having to reindex sections of the collection; for example, if a news event makes a previously unknown person famous, it would be nice to know if the collection contained pictures of that person. Finally, it is very hard to know what a picture is about:

> "the engineer requests an image of a misaligned mounting bracket . . . which only exists as an image described by a cataloguer as astronaut training . . ." [**?**]

These observations mean it would be pleasant to have automated tools that can describe and find pictures in large collections.

The first issue to think about in an application is what users want; how do they look for pictures, and what does it mean to describe a picture? Vision is difficult, so it is quite often very difficult to do what they want, but their needs should inform our thinking about what to do. For this problem, relatively little is known about what users need or want, or what there work practices are. We then discuss, briefly and at a high level, what tools might be able to do, prefatory to a more detailed discussion of current approaches to the problem.

24.1 Background

Information retrieval is the study of systems that recover texts from collections using various kinds of information. The topic is interesting to us because information retrieval researchers have become adept at performance analysis. Typically, the performance of information retrieval systems is described in terms of **recall** — the percentage of relevant items actually recovered — and **precision** — the percentage of recovered items actually relevant. Usually systems are assessed by plotting precision at various different levels of recall (obtained by varying match thresholds), and averaging these plots over “typical” queries. Bad experiments are easy to do, because it is often quite hard to tell what is relevant (i.e. what should have been recovered by a query), and even harder to tell how many relevant items appear in a large collection. Researchers are occasionally tempted to believe that good systems should have high recall and high precision. In fact, different applications can have sharply different requirements. High recall can be a serious problem. For example, consider a high recall search of the net for pictures of children; a huge percentage of the people who have both a personal web page and children have put pictures of their offspring on their web page, so the result will be a gigantic collection of pictures.

In many applications it is sufficient to find a few pictures satisfying the search criteria, that is low recall is not a problem. In the Enser study, for example, requesters are seldom burdened with more than 10 pictures, whatever the subject matter. As another example, consider filtering internet connections for offensive images; as long as a manager can be reasonably certain that any protracted flow of such images will result in an alarm — say 50% recall — the tool is usable. Usually, retrieving many incorrect pictures (low precision) is a greater problem because it will confuse and annoy a user. There *are* applications where high recall is essential: for example, a user searching for pictures that invalidate a patent will want to see every possibly relevant picture.

24.1.1 What do users want?

The most comprehensive study of the behaviour of users of image collections is Enser’s work on the then Hulton-Deutsch collection [**?**; **?**; **?**] (the collection has been acquired by a new owner since these papers were written, and is now known as the Hulton-Getty collection). This is a collection of prints, negatives, slides and the like, used mainly by media professionals. Enser studied the request forms on which client requests are logged; he classified requests into four semantic categories, depending on whether a unique instance of an object class is required or not and whether that instance is refined. Significant points include the fact that the specialised indexing language used gives only a "blunt pointer to regions of the Hulton collections" ([**?**], p. 35) and the broad and abstract semantics used to describe images. For example, users requested images of hangovers, physicists and the smoking of kippers. All these concepts are well beyond the reach of current image analysis techniques. As a result, there are few cases where one can obtain a tool that directly addresses a need. For the foreseeable future, the main constraint on the design of tools for finding images will be our quite limited understanding of vision.

However, useful tools can be built even with a limited understanding of vision (this extremely important point seems to be quite widely missed). It is hard to measure success. Enser suggests that the most reliable measure of the success of Hulton-Getty's indexing system is that the organisation is profitable. This test is a bit difficult to apply in practice, but there are a number of products available. IBM has produced a product for image search — QBIC (for Query By Image Content) — which has appeared in mass market advertising and appears to be successful. Similarly, Virage — a company whose main product is an image search engine — appears to be thriving (the company is described at [**?**]; a description of their technology appears in [**?**]).

The main source of value in any large collection is being able to find items, so that we can expect to see more search tools. Potential application areas include:

- **military intelligence:** vast quantities of satellite imagery of the globe exist, and typical queries involve finding militarily interesting changes — for example, concentrations of force — occuring at particular places (e.g. [**?**; **?**; **?**]).
- **planning and government:** satellite imagery can be used to measure development, changes in vegetation, regrowth after fires, etc. (e.g. [**?**]).
- **stock photo and stock footage:** commercial libraries — which often have extremely large and very diverse collections — sell the right to use particular images (e.g. [**?**; **?**; **?**]).
- **access to museums:** museums are increasingly releasing collections, typically at restricted resolutions, to entice viewers into visiting the museum (e.g. [**?**; **?**; **?**]).
- **trademark enforcement:** as electronic commerce grows, so does the opportunity for automatic searches to find violations of trademark (e.g. [**?**; **?**; **?**; **?**; **?**; **?**]).
- **indexing the web:** indexing web pages appears to be a profitable activity. Users may also wish to have tools that allow them to avoid offensive images or

Missing
Figure

Figure 24.1. *Iconic matching systems look for images that have exactly the same pixels as the image sought. Images can be matched very quickly using this criterion if they are appropriately coded. However, there are relatively few applications where this criterion applies, because it requires the user to have a fairly precise notion of what the picture sought actually looks like — in this example, the user would need to know how Scheile's model had posed.* figure from the paper "Fast Multiresolution Image Querying", Jacobs, Finkelstein and Salesin, SIGGRAPH-95, p unknown, in the fervent hope that permission will be granted

advertising. A number of tools have been built to support searches for images on the web using techniques described below (e.g. [**?**; **?**; **?**]).

- **medical information systems:** recovering medical images "similar" to a given query example might give more information on which to base a diagnosis or to conduct epidemiological studies (e.g. [**?**; **?**]).

24.1.2 What can tools do?

There are three ways to look for an image: one can search for an exact match, for an image that "looks similar", or for an image with particular object-level semantics.

In **iconic matching**, we are seeking an image that looks as much like an example picture — which we might draw, or supply — as possible. The ideal match would have exactly the same pixel value in each location. This might be useful for users who have a clear memory of images that appear in the collection. The best known system of this form is due to Jacobs *et al.* of the University of Washington [**?**], illustrated in figure 24.1. Iconic matching tends to be used relatively seldom, because it is usually too difficult to remember what the image being sought looks like.

Appearance: In some applications — for example, finding trademark infringements — the structure of the whole image is important. In these applications, we think of the image as an arrangement of coloured pixels, rather than a picture of objects. This abstraction is often called **appearance**. The distinction between appearance and object semantics is somewhat empty — how do we know what's in an image except by its appearance? — but the approach is very important in practice, because it is quite easy to match appearance automatically. Appearance is particularly helpful when the *composition* of the image is important. For example, one could search for stock photos using a combination of appearance cues and keywords, and require the user to exclude images with the right composition but the wrong semantics. The central technical problem in building a tool that searches on appearance is defining a useful notion of image similarity; section 24.2 illustrates a variety of different strategies.

Object level semantics: It is very difficult to cope with high-level semantic queries ("a picture of the Pope, kissing a baby") using appearance or browsing tools. Finding tools use elements of the currently limited understanding of object recognition to help a user query for images based on this kind of semantics, at a variety of levels. It is not known how to build finding tools that can handle high-level semantic queries, nor how to build a user interface for a general finding tool; nonetheless, current technology can produce quite useful tools for various special cases (section 24.3).

Browsing

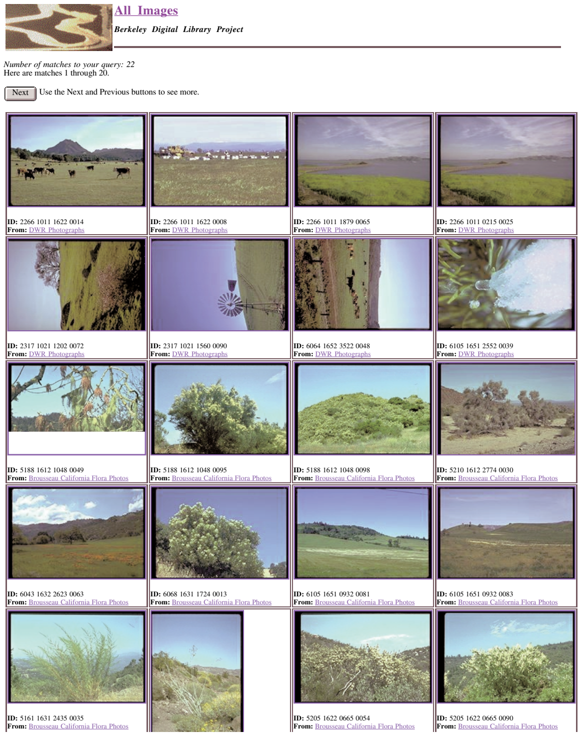
Searching for images raises some very difficult problems (for example, assume you had a perfect object recognition system; how would you describe the picture you wanted? as [**?**; **?**; **?**] show, using human indexers and language doesn't seem to work even close to perfectly). This means that tools tend to be quite erratic in practice. Fortunately, image collections are often highly correlated (usually as a result of the way they are created).

This correlated structure can be exploited if we provide a tool for browsing. A user then searches with the search tool, and can choose to look at items "near" to any hits returned by the search tool. Ideally, a browsing tools will display images that are "similar" — this could mean that they look similar, or have similar appearance, or lie close to one another in the collection, etc. — in a way that makes their similarity apparent, and provide some form of interaction that makes it possible to move through the collection in different "directions".

Building useful browsing tools also requires an effective notion of image similarity. Constructing a good user interface for such systems is difficult; desirable features include a clear and simple query specification process, and a clear presentation of the internal representation used by the program, so that failures are not excessively puzzling. Typically, users are expected to offer an example image or to fill in a form-based interface to search for the first image, and then can move around the collection by clicking on samples offered by the browsing tool.

24.2 Appearance

Images are often highly stylised, particularly when the intent of the artist is to emphasize a particular object or a mood. This means that the overall layout of an image can be a guide to what it depicts, so that useful query mechanisms can be built by looking for images that "look similar" to a sample image, a sketched sample, or textual specification of appearance. The success of such methods rests on the sense in which images look similar. It is important to convey to the user the sense in which images look similar, because otherwise mildly annoying errors can become extremely puzzling. A good notion of similarity is also important for efficient browsing, because a user interface that can tell how different images are, can lay out a display of images to suggests the overall structure of the section of the collection being displayed. We will concentrate on discussing appearance matching rather than browsing, because the technical issues are so similar.



All Images

Berkeley Digital Library Project

Number of matches to your query: 22
Here are matches 1 through 20.

Next Use the Next and Previous buttons to see more.

ID: 2266 1011 1622 0014
From: DWR Photographs

ID: 2266 1011 1622 0008
From: DWR Photographs

ID: 2266 1011 1879 0065
From: DWR Photographs

ID: 2266 1011 0215 0025
From: DWR Photographs

ID: 2317 1021 1202 0072
From: DWR Photographs

ID: 2317 1021 1560 0090
From: DWR Photographs

ID: 6064 1652 3522 0048
From: DWR Photographs

ID: 6105 1651 2552 0039
From: DWR Photographs

ID: 5188 1612 1048 0049
From: Brousseau California Flora Photos

ID: 5188 1612 1048 0095
From: Brousseau California Flora Photos

ID: 5188 1612 1048 0098
From: Brousseau California Flora Photos

ID: 5210 1612 2774 0030
From: Brousseau California Flora Photos

ID: 6043 1632 2623 0063
From: Brousseau California Flora Photos

ID: 6068 1631 1724 0013
From: Brousseau California Flora Photos

ID: 6105 1651 0932 0081
From: Brousseau California Flora Photos

ID: 6105 1651 0932 0083
From: Brousseau California Flora Photos

ID: 5161 1631 2435 0035
From: Brousseau California Flora Photos

ID: 5205 1622 0665 0054
From: Brousseau California Flora Photos

ID: 5205 1622 0665 0090
From: Brousseau California Flora Photos

Figure 24.2. *Results from a query to the Calphotos collection that sought pastoral scenes, composed by searching for images that contain many green and light blue pixels. As the results suggest, such colour histogram queries can be quite effective.*

24.2.1 Histograms and correlograms

A popular measurement of similarity compares counts of the number of pixels in particular colour categories. For example, a sunset scene and a pastoral scene would be very different by this measure, because the sunset scene contains many red, orange and yellow pixels and the pastoral scen will have a preponderance of green (grass), blue (sky) and perhaps white (cloud) pixels (e.g. figure 24.2). Furthermore, sunset scenes will tend to be similar; all will have many red, orange and yellow pixels and few others.

A **colour histogram** is a record of the number of pixels in an image or a region that fall into particular quantization buckets in some colour space (RGB is popular, for reasons we cannot explain). If the colour histogram for an image of an object fits into the histogram for an image (in the sense illustrated in figure **??**), then it is possible that that object is present in the image — if the illumination is not expected to vary all that much. This test can be quite sensitive to viewing direction and scale, because the relative number of pixels of a given colour can change sharply. Nonetheless, it has the advantage of being quick and easy, and applies to things like clothing which may have bright colours but little or no recognisable shape.

Colour histogram matching has been extremely popular; it dates back at least to the work of Swain and Ballard [**?**], and has been used in a number of systems used in practice [**?**; **?**; **?**]. The usefulness of colour histograms is slightly surprising, given how much image information the representation discards; for example, Chappelle *et al.* at ATT have shown that images from the Corel collection[1] can be classified by their category in the collection using colour histogram information alone [**?**].

There is no record in a colour histogram of *where* coloured pixels are with respect to one another. Thus, for example, pictures of the French and UK flags are extremely similar according to a colour histogram measure — each has red, blue and white pixels in about the same number; it is the spatial layout of the pixels that differs. One problem that can result is that pictures taken from slightly different viewing positions look substantially different by a colour histogram measure (figure 24.3). This effect can be alleviated by considering the probability that a pixel of some colour lies within a particular pixel of another colour (which can be measured by counting the number of pixels at various distances). For small movements of the camera, these probabilities will be largely unchanged, so that similarity between these **colour correlograms** yields a measure of similarity between images. Requiring that colour correlograms be similar provides another measure of image similarity. The computational details — which have been worked out by Zabih and colleagues [**?**; **?**].

24.2.2 Textures and textures of textures

Colour histograms contain no information about the layout of colour pixels. An explicit record of layout is the next step. For example, a snowy mountain image will have bluer regions on top, whiter regions in the middle, then a bluer region at the bottom (the lake at the foot of the mountain), whereas a waterfall image will have a darker region on the left and right and lighter vertical stripe in the center. These layout templates were introduced by Lipson, Grimson and Sinha at MIT; they can be learned for a range of images, and appear to provide a significant improvement over a colour histogram [**?**].

Looking at image texture is a natural next step, because texture is the difference between, say, a field of flowers (many small orange blobs) and a single flower (one

---

[1] A collection of 60, 000 images quite commonly used in vision research; available in three series from the Corel corporation, ********

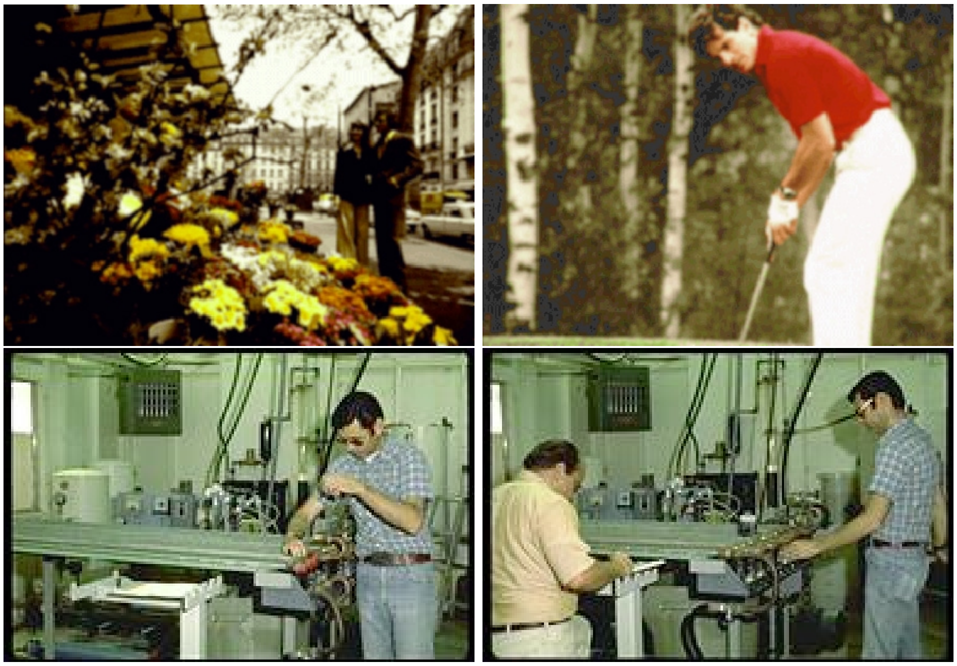


Figure 24.3. *The top two figures have the same colour histogram; the red patch on the golfers shirt appears in the other image as brown looking flowers. These flowers are redder than they look (their hue is changed somewhat by the fact that they are small, and don't contrast strongly with their background), although not quite as red as the shirt. However, in the scheme of colour categories used the colours are regarded as equivalent. The bottom two figures have similar content, but quite different colour histograms; the person in the peach shirt appears in only one figure, and the person in the blue occupies more space in one than in the other. However, if one looks at a representation of the extent to which pixels of a given colour lie near pixels of some other colour, the pictures are quite similar — many blue pixels lie near either blue pixels, green ones or white ones. This information is captured by the colour correlogram. More information on colour correlograms can be found in [***?***; ***?***]; picture by kind permission of R. Zabih.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

big orange blob), or a dalmation and a zebra. Most people know texture when they see it, though the concept is either difficult or impossible to define. Typically, textures are thought of as spatial arrangements of small patterns — for example, a tartan is an arrangement of small squares and lines, and the texture of a grassy field is an arrangement of thin bars.

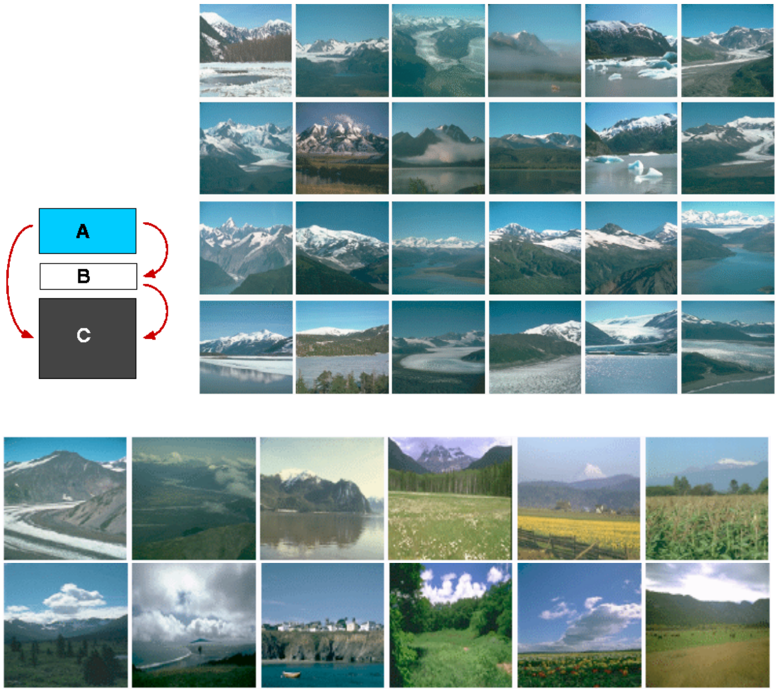
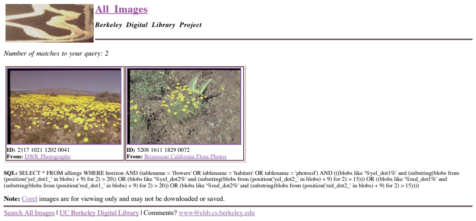


Figure 24.4. *Spatial layout of coloured regions is a natural guide to the content of many types of image. The figure on the* **top left** *shows a layout of coloured regions that suggests a scene showing snowy mountains;* **top right**, *the figures recovered by this criterion that actually do show snowy mountains;* **center**, *views of mountains that were in the collection but not recovered and* **bottom**, *images that meet the criterion but do not actually show a view of a snowy mountain. More detail appears in [**?**]; figure by kind permission of W.E.L. Grimson and P. Lipson.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

The usual strategy for finding these subpatterns is to apply a linear filter to the image (see chapters **??** and **??**), where the kernel of the filter looks similar to the pattern element. From filter theory, we have that strong responses from these filters suggest the presence of the particular pattern; several different filters can be applied, and the statistics of the responses in different places then yield a decomposition of the picture into spotty regions, barred regions, and the like [**?**; **?**; **?**].

A histogram of filter responses is a first possible description of texture. For example, one might query for images with few small yellow blobs. This mechanism is used quite successfully in the Calphotos collection at Berkeley (`http://elib.cs.berkeley.edu/photos`; there are many thousands of images of California natural resources, flowers and


All Images
Berkeley Digital Library Project

Number of matches to your query: 2

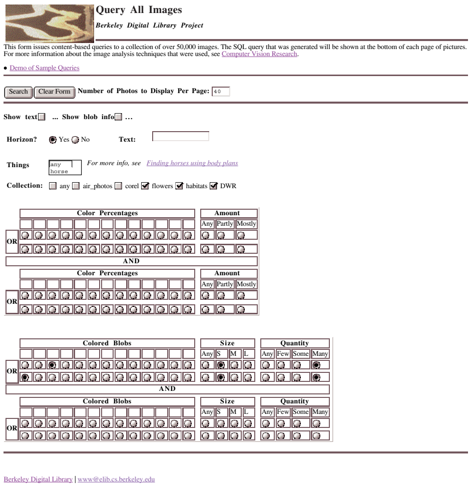
ID: 2317 1021 1202 0041
From: DWR Photographs

ID: 5208 1611 1829 0072
From: Brousseau California Flora Photos

SQL: SELECT * FROM allimgs WHERE horizon AND (tablename = 'flowers' OR tablename = 'habitats' OR tablename = 'photocd') AND (((blobs like '%yel_dot1%' and (substring(blobs from (position('yel_dot1_' in blobs) + 9) for 2) > 20)) OR (blobs like '%yel_dot2%' and (substring(blobs from (position('yel_dot2_' in blobs) + 9) for 2) > 15))) OR ((blobs like '%red_dot1%' and (substring(blobs from (position('red_dot1_' in blobs) + 9) for 2) > 20)) OR (blobs like '%red_dot2%' and (substring(blobs from (position('red_dot2_' in blobs) + 9) for 2) > 15))))

Note: Corel images are for viewing only and may not be downloaded or saved.

Search All Images | UC Berkeley Digital Library | Comments? www@elib.cs.berkeley.edu


Query All Images
Berkeley Digital Library Project

This form issues content-based queries to a collection of over 50,000 images. The SQL query that was generated will be shown at the bottom of each page of pictures. For more information about the image analysis techniques that were used, see Computer Vision Research.

- Demo of Sample Queries

Search
Clear Form

Number of Photos to Display Per Page: 40

Show text ☐ ... **Show blob info** ☐ ...

Horizon? ● Yes ○ No **Text:**

Things any horse *For more info, see* Finding horses using body plans

Collection: ☐ any ☐ air_photos ☐ corel ☑ flowers ☑ habitats ☑ DWR

Color Percentages
Amount
Any
Partly
Mostly
OR
AND
Color Percentages
Amount
Any
Partly
Mostly
OR
Colored Blobs
Size
Any
S
M
L
Quantity
Any
Few
Some
Many
OR
AND
Colored Blobs
Size
Any
S
M
L
Quantity
Any
Few
Some
Many
OR
Berkeley Digital Library | www@elib.cs.berkeley.edu

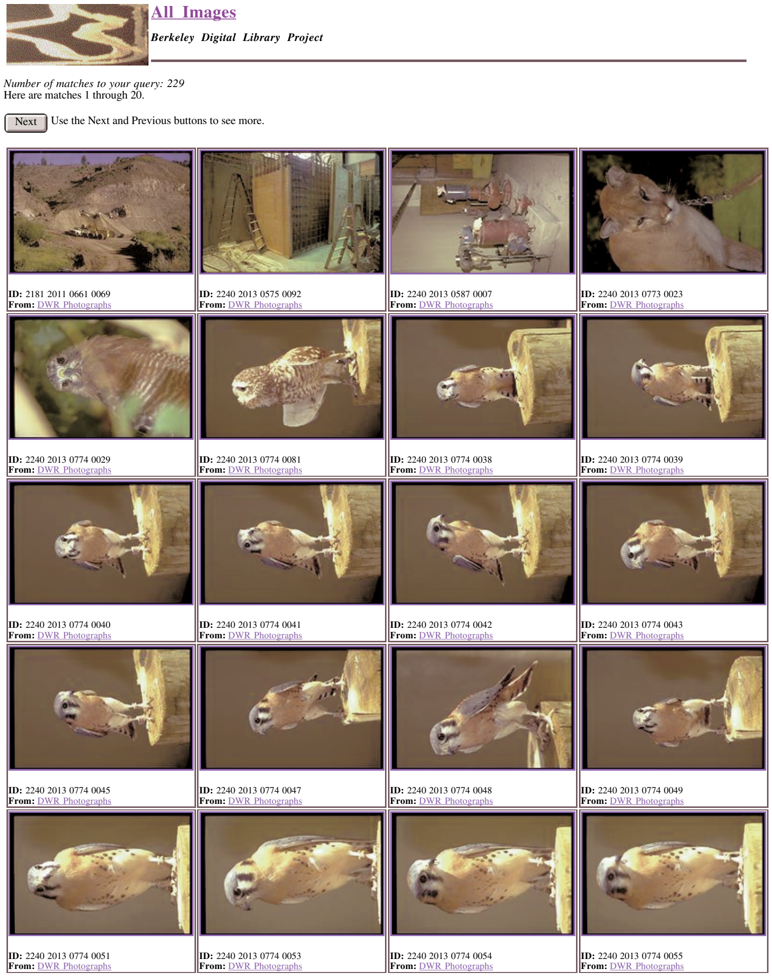
Figure 24.5. Left: *Specifying a query to the Calphotos collection using colour and texture information. I have selected images that have a horizon, and some red or yellow blobs, with the intention of finding views of fields of flowers; results appear in figure 24.5.* **Right:** *Images obtained from the Calphotos collection using the query of figure 24.5.*

wildlife). As figures 24.5-24.6 illustrate, a combination of colour and blob queries can be used to find quite complex images.

Texture histograms have some problems with camera motion; as the person in figure 24.3 approaches the camera, the checks on his shirt get bigger in the image. The pattern of texture responses could change quite substantially as a result. This is another manifestation of a problem we saw earlier with colour histograms; the size of a region spanned by an object changes as the camera moves closer to or further from the object.

A strategy for minimizing the impact of this effect is to define a family of allowable transformations on the image — for example, scaling the image by a factor in some range. We now apply each of these transformations, and measure the similarity between two images as the smallest difference that can be obtained using a transformation. For example, we could scale one image by each legal factor and look for the smallest difference between colour and texture histograms. This **earth-movers distance** — due to Rubner, Tomasi and Guibas at Stanford — allows a wide variety of transformations; furthermore, in [**?**], it has been coupled with a process for laying out images that makes the distance between images in the display reflect the dissimilarity between images in the collection. This approach allows for rapid and intuitive browsing (figure 24.7).

The spatial layout of textures is a powerful cue. For example, in aerial images, housing developments have a fairly characteristic texture, and the layout of this



Number of matches to your query: 229
Here are matches 1 through 20.

Use the Next and Previous buttons to see more.

ID: 2181 2011 0661 0069 **From:** DWR Photographs	**ID:** 2240 2013 0575 0092 **From:** DWR Photographs	**ID:** 2240 2013 0587 0007 **From:** DWR Photographs	**ID:** 2240 2013 0773 0023 **From:** DWR Photographs
ID: 2240 2013 0774 0029 **From:** DWR Photographs	**ID:** 2240 2013 0774 0081 **From:** DWR Photographs	**ID:** 2240 2013 0774 0038 **From:** DWR Photographs	**ID:** 2240 2013 0774 0039 **From:** DWR Photographs
ID: 2240 2013 0774 0040 **From:** DWR Photographs	**ID:** 2240 2013 0774 0041 **From:** DWR Photographs	**ID:** 2240 2013 0774 0042 **From:** DWR Photographs	**ID:** 2240 2013 0774 0043 **From:** DWR Photographs
ID: 2240 2013 0774 0045 **From:** DWR Photographs	**ID:** 2240 2013 0774 0047 **From:** DWR Photographs	**ID:** 2240 2013 0774 0048 **From:** DWR Photographs	**ID:** 2240 2013 0774 0049 **From:** DWR Photographs
ID: 2240 2013 0774 0051 **From:** DWR Photographs	**ID:** 2240 2013 0774 0053 **From:** DWR Photographs	**ID:** 2240 2013 0774 0054 **From:** DWR Photographs	**ID:** 2240 2013 0774 0055 **From:** DWR Photographs

Figure 24.6. *Response images obtained by querying the Calphotos collection for images with at least one large brown blob, more than one small black blobs, and some green; this query is intended to find animals or birds.*

texture gives cues to the region sought. In the Netra system, built by Ma and Manjunath at U.C. Santa Barbara, textures are classified into into stylised families (yielding a "texture thesaurus") which are used to segment very large aerial images; this approach exploits the fact that, while there is a very large family of possible textures, only some texture distinctions are significant. Users can then use example regions to query a collection for similar views; for example, obtaining aerial pictures of a particular region at a different time or date to keep track of such matters as

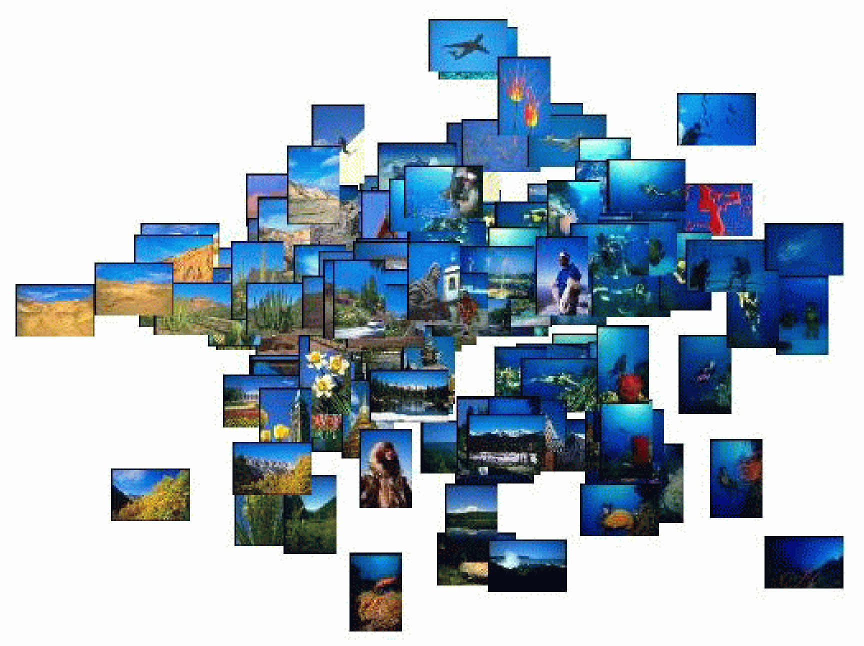


Figure 24.7. *Images laid out according to their similarity using the earth mover's distance (EMD). The EMD can be computed very fast so that displays like this — where distances between images on the display reflect the EMDs between them as faithfully as possible — can be created online. Large numbers of pictures returned from a query into an image database can thus be viewed at a glance, and a mouse click in the neighborhood of pictures that look similar to what the user is looking for tells the retrieval system where to search next. With this technology, users browse and navigate in an image database, just as they would browse through a department store. Because of the large number of images displayed, and their spatially intuitive layout, users quickly form a mental model of what is in the database, and rapidly learn where to find the pictures they need. More information on the EMD can be found in [**?**]; figure by kind permission of C. Tomasi.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

the progress of development, traffic patterns, or vegetation growth (figure 24.8; [**?**; **?**; **?**; **?**]).

Regions of texture responses form patterns, too. For example, if an image shows a pedestrian in a spotted shirt, then there will be many strong responses from spot detecting filters; the region of strong responses will look roughly like a large bar. A group of pedestrians in spotted shirts will look like a family of bars, which is itself a texture. These observations suggest applying texture finding filters to the outputs

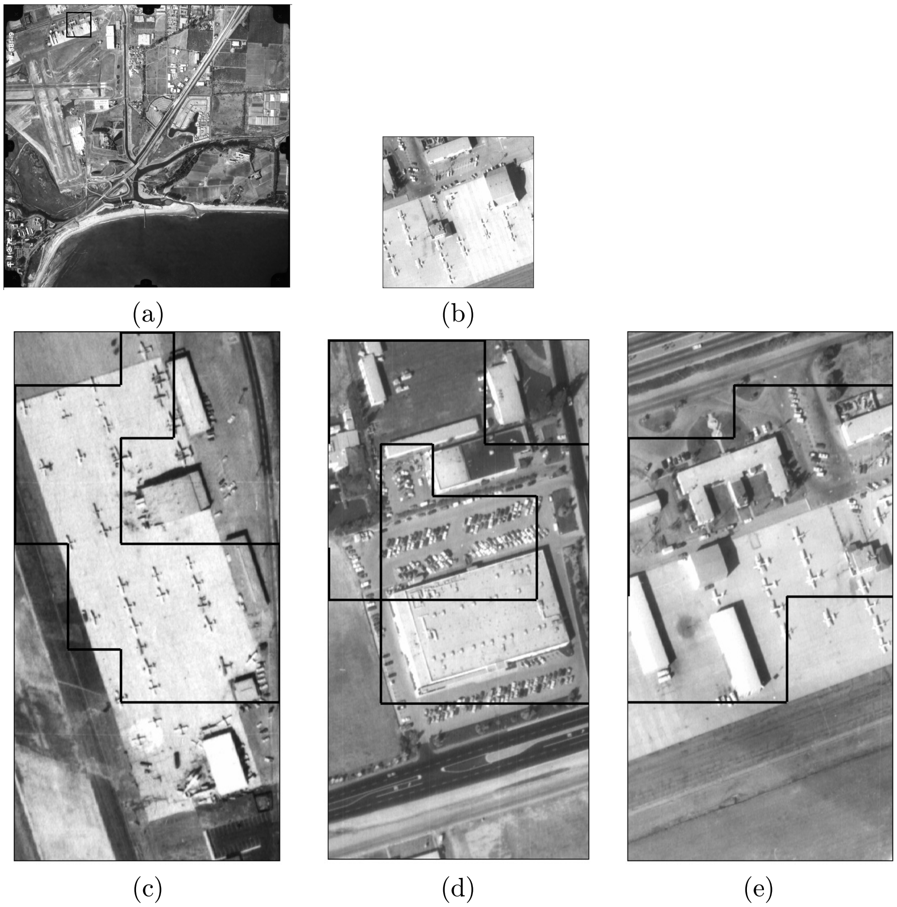


Figure 24.8. *A texture-based search in an aerial image. (a) shows the down-sampled version of the aerial photograph from which the query is derived. (b) shows a full-resolution detail of the region used for the query. The region contains aircraft, cars and buildings. (c)-(e) show the ordered three best results of the query. Once again, the results come from three different aerial photographs. This time, the second and third results are from the same year (1972) as the query photograph but the first match is from a different year (1966). More details appear in [**?**]; figure by kind permission of B.S. Manjunath.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

of texture finding filters — perhaps recurring several times — and using measures of similarity of these responses as a measure of image similarity. This approach — due to DeBonet and Viola at MIT — involves a large number of features, so it is impractical to ask users to fill in a form. Instead, as in figure 24.9 and figure 24.10,

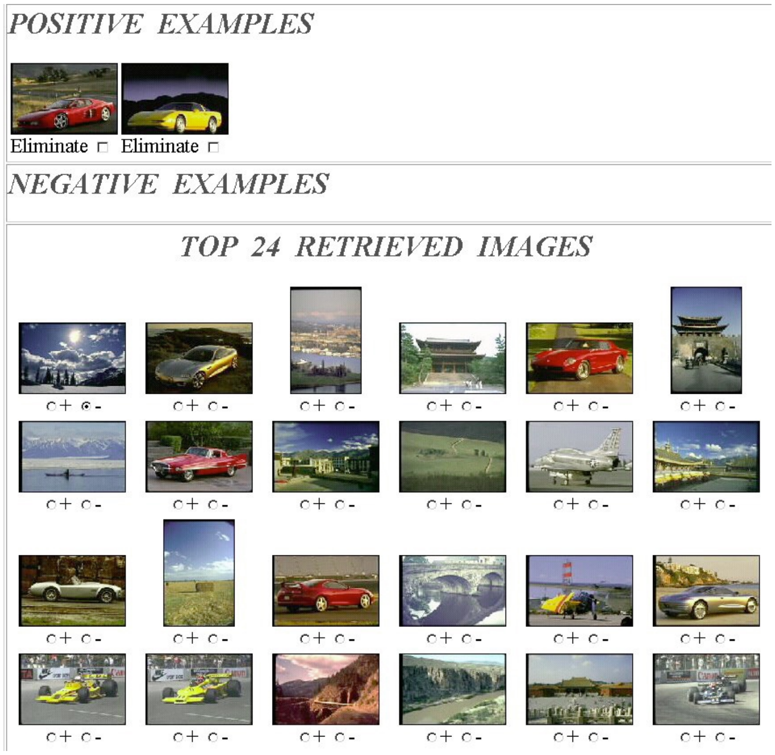


Figure 24.9. *Querying using the "texture of textures" approach. The user has identified two pictures of cars as positive examples; these would respond strongly to large horizontal bar filters, among others. This query results in a number of returned images, containing several images of cars; figure 24.10 shows the effect of refining this query by exhibiting negative examples. Figure by kind permission of P. Viola.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

the authors use an approach where users select positive and negative example images, and the system searches for images that are similar to the positive examples and dissimilar to the negative examples [**?**].

24.3 Finding

The distinction between appearance tools and finding tools is somewhat artificial; we can tell what objects are based on their differences in appearance. The tools described in this section try to estimate object-level semantics more or less directly. Such systems must first *segment* the image — i.e. decide which pixels lie on the object of interest. *Template matching* systems then look for characteristic patterns

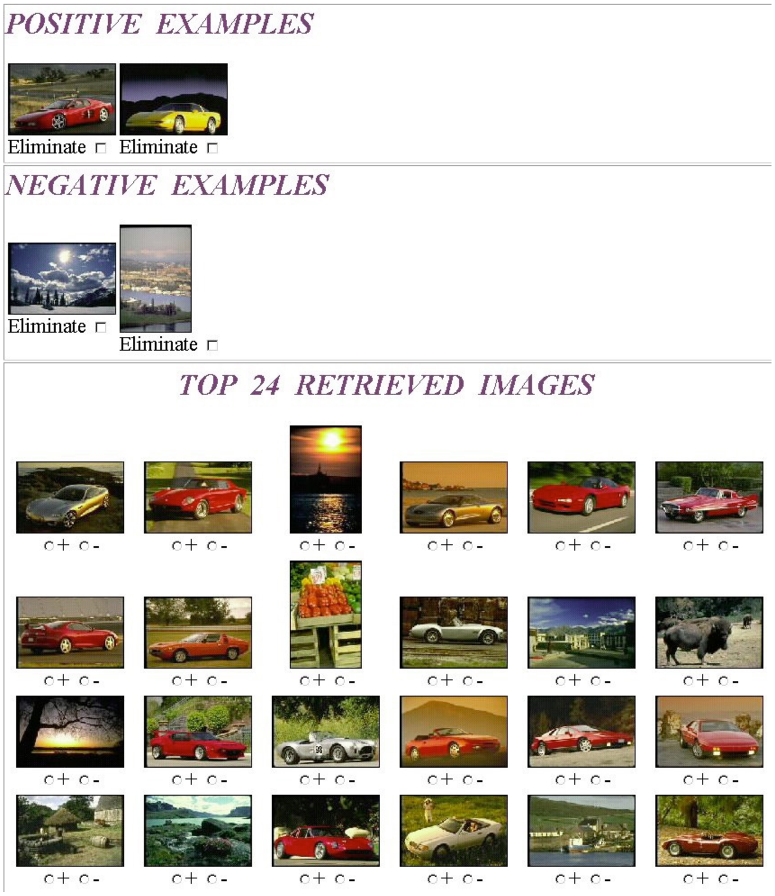
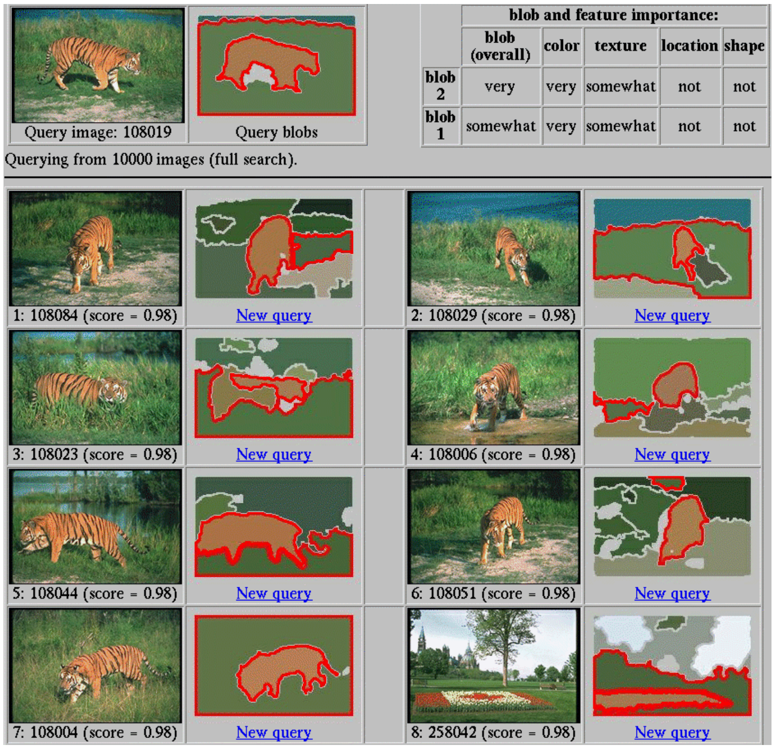


Figure 24.10. *Querying using the "texture of textures" approach. The query has been refined by providing some negative examples, yielding a response set that contains more car images. More detail on this approach appears in [?]; figure by kind permission of P. Viola.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

associated with particular objects; finally, *correspondence* reasoning can be used to identify objects using spatial relationships between parts.

Structure in a collection is helpful in finding semantics, because it can be used to guide the choice of particular search mechanisms. Photobook — due to Pentland, Picard and Sclaroff at MIT — is a system that provides three main search categories: shape Photobook searches for isolated objects (for example, tools or fishes) using contour shape measured as elastic deformations of a contour; appearance Photobook can find faces using a small number of principal components; and texture Photobook



	blob and feature importance:				
	blob (overall)	color	texture	location	shape
blob 2	very	very	somewhat	not	not
blob 1	somewhat	very	somewhat	not	not

Querying from 10000 images (full search).

Query image: 108019
Query blobs
1: 108084 (score = 0.98)
New query
2: 108029 (score = 0.98)
New query
3: 108023 (score = 0.98)
New query
4: 108006 (score = 0.98)
New query
5: 108044 (score = 0.98)
New query
6: 108051 (score = 0.98)
New query
7: 108004 (score = 0.98)
New query
8: 258042 (score = 0.98)
New query

Figure 24.11. *Blobworld query for tiger images. Users of image databases generally want to find images containing particular objects, not images with particular global statistics. The Blobworld representation facilitates such queries by representing each image as a collection of regions (or "blobs") which correspond to objects or parts of objects. The image is segmented into regions automatically, and each region's color, texture, and shape characteristics are encoded. The user constructs a query by selecting regions of interest. The Blobworld version of each retrieved image is shown, with matching regions highlighted; displaying the system's internal representation in this way makes the query results more understandable and aids the user in creating and refining queries. Experiments show that queries for distinctive objects such as tigers and cheetahs have much higher precision using the Blobworld system than using a similar system based only on global color and texture descriptions. Blobworld is described in greater detail in [**?**; **?**]; figure by kind permission of C. Carson.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

uses a texture representation to find textured swatches of material [**?**]

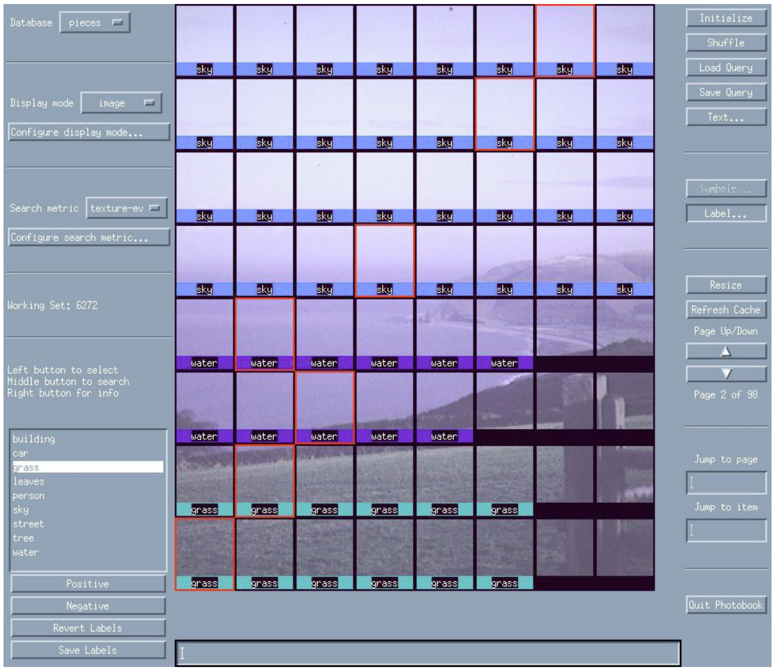
24.3.1 Annotation and segmentation

A natural step in determining image semantics is to classify the type of material image patches represent; for example, “sky”, “buildings”, etc., as opposed to “blue”, “grey”. Generally, this kind of classification would need to be done with a mixture of user input (to establish appropriate categories and provide examples from those categories) and automatic annotation (for speed and efficiency). A combination of colour and texture features is often, but not always, distinctive of a region; a particular difficulty is knowing which features to use and which to ignore in classifying an image patch. For example, telling sky from grass involves looking at colour; telling concrete from sky may require ignoring colour and emphasizing texture. Foureyes — due to Minka and Picard at MIT — uses techniques from machine learning to infer appropriate features from user annotation practices, using across-image groupings (which patches have been classified as “sky” in the past?) and in-image groupings (which patches are classified as “sky” in this image?) [**?**; **?**; **?**]. As a result, a user annotating an image can benefit from past experience, as illustrated in figure 24.12.

Humans decompose images into pieces corresponding to the objects we are interested in, and classification is one way to achieve this *segmentation.* Segmentation is a crucial idea, because it means that irrelevant information can be discarded in comparing images. For example, if we are searching for an image of a tiger, it should not matter whether the background is snow or grass; the tiger is the issue. However, if the whole image is used to generate measures of similarity, a tiger on grass will look very different from a tiger on snow. These observations suggest segmenting an image into regions of pixels that belong together in an appropriate sense, and then allowing the user to search on the properties of particular regions. The most natural sense in which pixels belong together is that they come from a single object; currently, it is almost never possible to use this criterion, because we don’t know how to tell when this is the case. However, objects usually result in image regions of coherent colour and texture, so that pixels that belong to the same region have a good prospect of belonging to an object.

VisualSEEK — due to Smith and Chang at Columbia — automatically breaks images into regions of coherent colour, and allows users to query on the spatial layout and extent of coloured regions. Thus, a query for a sunset image might specify an orange background with a yellow blob lying on that background [**?**].

Blobworld is a system built at Berkeley by Carson *et al.* that represents images in terms of a collection of regions of coherent colour and texture [**?**; **?**; **?**; **?**]. The representation is displayed to the user, with region colour and texture displayed inside elliptical blobs, which represent the shape of the image regions. The shape of these regions is represented crudely, because details of the region boundaries are not cogent. A user can query the system by specifying which blobs in an example image are important, and what spatial relations should hold (figure 24.11).



Database pieces

Display mode image

Search metric texture-ev

Working Set: 6272

Left button to select
Middle button to search
Right button for info

Page 2 of 98

Configure display mode...
Configure search metric...
building
car
grass
leaves
person
sky
street
tree
water
Positive
Negative
Revert Labels
Save Labels
sky
water
grass
Initialize
Shuffle
Load Query
Save Query
Text...
Symbols...
Label...
Resize
Refresh Cache
Page Up/Down
Jump to page
Jump to item
Quit Photobook

Figure 24.12. *An image annotated with Foureyes; red patches of image have been explicitly labelled "sky", "grass" or "water". Other labels are inferred by Foureyes from previous annotations and from the information given in these examples. More information appears in [?]; figure by kind permission of R. Picard.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

24.3.2 Template matching

Some objects have quite characteristic appearance for a wide range of viewing directions and conditions. Template matching is an object recognition strategy that finds objects by matching image patches with example templates (chapter **??**). A natural application of template matching is to construct whole-image templates that correspond to particular semantic categories (figure 24.13 and [**?**]). These templates can be constructed off-line, and used to simplify querying by allowing a user to use an existing template, rather than compose a query.

Face finding is a particularly good case for template matching. Frontal views of faces are extremely similar, particularly when the face is viewed at low resolution — the main features are then a dark bar at the mouth, dark blobs where the eyes are, and lighter patches at the forehead, nose and mouth. This means that faces can be found, independent of the identity of the person, by looking for this pattern. Typical face finding systems extract small image windows of a fixed size,

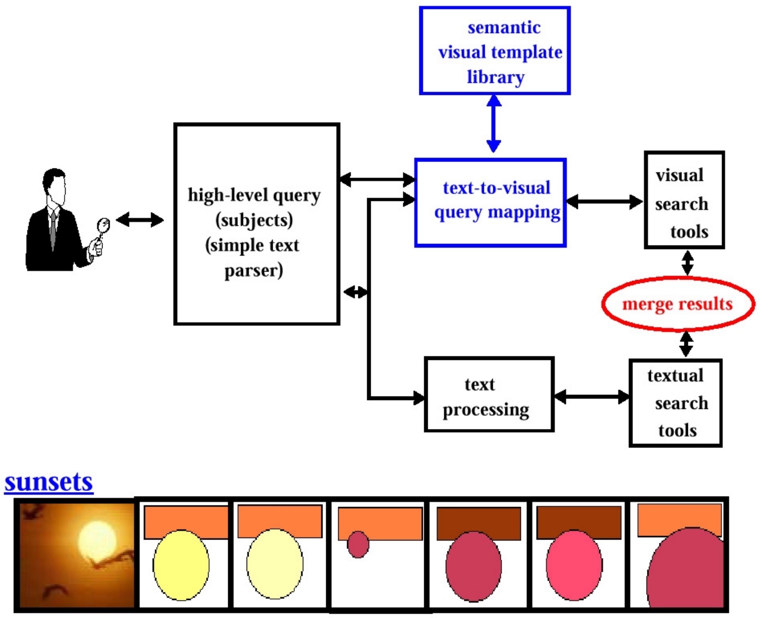

semantic visual template library
high-level query (subjects) (simple text parser)
text-to-visual query mapping
visual search tools
merge results
text processing
textual search tools
sunsets

Figure 24.13. *To remove the burden of drawing detailed low-level sketches from users, the Semantic Visual Template system helps users to develop personalized search templates. The library of semantic templates can then be used to assist users in high-level multi-media query. The top figure shows the overall structure of a system using semantic templates, and the lower figure shows a template for sunset images. More details appear in [?]; figure by kind permission of S-F. Chang.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

prune these windows to be oval, correct for lighting across the window, and then use a learned classifier to tell whether a face is present in the window [?; ?; ?; ?]. Our figure (figure 24.14) illustrates work along these lines due to Rowley, Baluja and Kanade at CMU; a similar approach has been used successfully by Sung and Poggio at MIT. This process works for both large and small faces, because windows are extracted from images at a variety of resolutions (windows from low resolution images yield large faces, and those from high resolution images yield small faces). Because the pattern changes when the face is tilted to the side, this tilt must be estimated and corrected for; this is done using a mechanism learned from data [?]. Knowing where the faces are is extremely useful, because many natural queries refer to the people present in an image or a video.

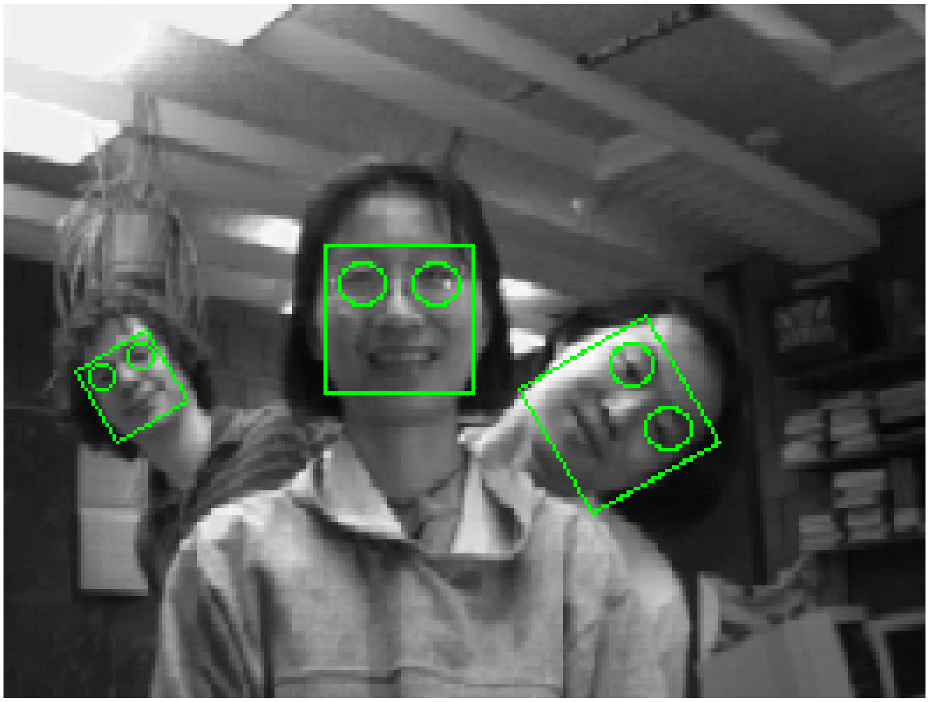


Figure 24.14. *Faces found by a learned template matching approach; the eyeholes in the green mask are to indicate the orientation of the face. More details appear in [**?**; **?**; **?**]; figure by kind permission of T. Kanade.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

24.3.3 Shape and correspondence

If object appearance can vary, template matching becomes more difficult as one is forced to adopt many more templates. There is a good template matching system for finding pedestrians, which appears to work because pedestrians tend to be seen at low resolution with their arms at their sides [**?**]. However, building a template matching system to find people is intractable, because clothing and configuration can vary too widely. The general strategy for dealing with this difficulty is to look for smaller templates — perhaps corresponding to "parts" — and then look for legal configurations.

One version of this technique involves finding "interest points" — points where combinations of measurements of intensity and its derivatives take on unusual values (for example, at corners). As Schmid and Mohr of INRIA and Zisserman of Oxford have shown, the spatial arrangement of these points is quite distinctive in many cases. For example (as figure 24.15 illustrates) the arrangement of interest points in

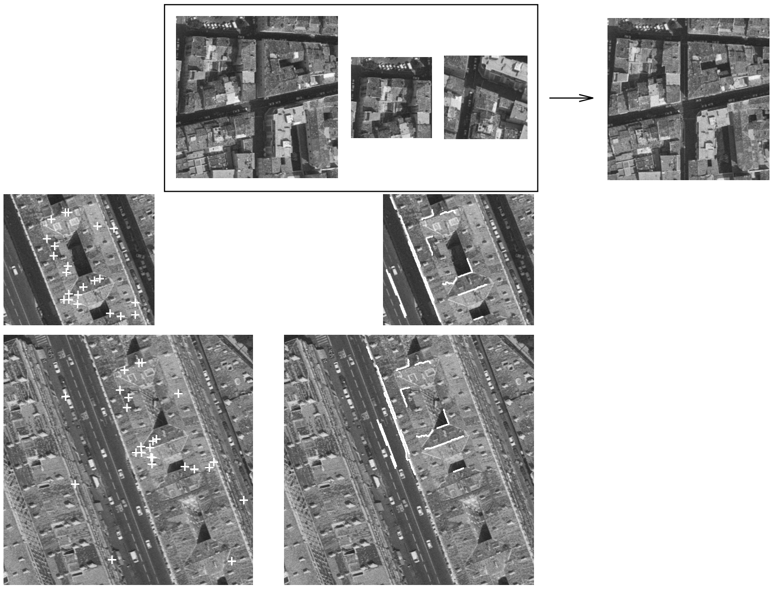


Figure 24.15. *Images can be queried by detecting "interest points" on the image and then matching on configurations of these points, based on their geometric distribution and the grey level pattern surrounding each point. The matching process is very efficient (it uses 'indexing'), and is tolerant of missing points in the configuration. In the example shown here, the image on the top right can be correctly retrieved from a collection of paintings, aerial images and images of 3D objects using any of the images on the top left. Interest points used during the matching process, shown in white, for a query image (small inset on left) and the best match (bottom left). Additional evidence is obtained from the image-image transformation to confirm the match is correct; on the right, edges which match under this transformation in the query (inset) and the result (bottom right). Notice that the two images have been taken from different viewpoints so that the building's shape differs between images. Also the scenes are not identical because cars have moved. Further details are given in [**?**]; figure by kind permission of A. Zisserman.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

an aerial view of Marseille is unaffected by the presence of cars; this means that one can recover and register aerial images of the same region taken at different times of day using this technique. Furthemore, once interest points have been matched, an image-image transformation is known, which can be used to register the images. Registration yields further evidence to support the match, and can be used to compare, say, traffic by comparing the two images at specific points.

This form of correspondence reasoning extends to matching image components with object parts at a more abstract level. People and many animals can be thought of as assemblies of cylinders (corresponding to body segments). A natural finding representation uses grouping stages assemble image components that could correspond to appropriate body segments or other components.

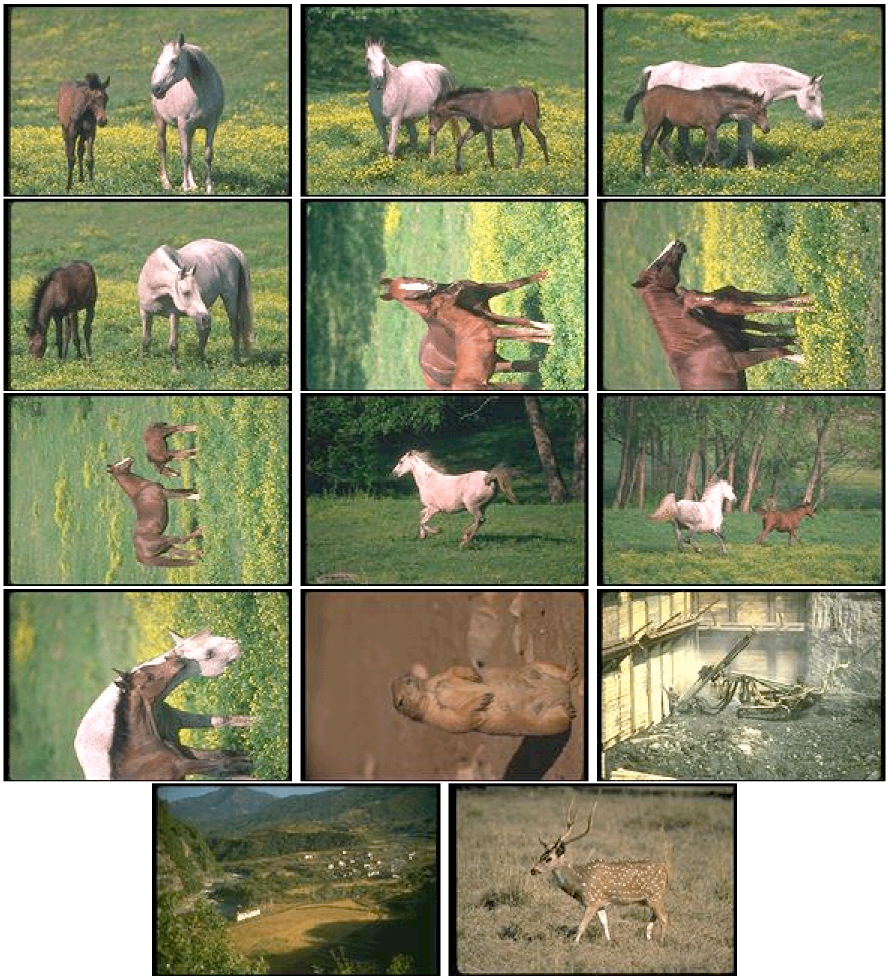


Figure 24.16. *Images of horses recovered using a body plan representation, from a test collection consisting of 100 images containing horses, and 1086 control images with widely varying content. Note that the method is relatively insensitive to aspect, but can be fooled by brown, horse-shaped regions. More details appear in [**?**].*

Forsyth of U.C. Berkeley and Fleck of HP Labs have used this representation for two cases; the first example identifies pictures containing people wearing little or no clothing. This is an interesting example: firstly, it is much easier than finding clothed people, because skin displays very little variation in colour and texture in images, whereas the appearance of clothing varies very widely; secondly, many people are interested in avoiding or finding images based on whether they contain unclad people. This program has been tested on an usually large and unusually diverse set of images; on a test collection of 565 images known to contain lightly clad people and 4289 control images with widely varying content, one tuning of the program marked 241 test images and 182 control images (more detailed information appears in [**?**; **?**]). The second example used a representation whose combinatorial structure — the order in which tests were applied — was built by hand, but where the tests were learned from data. This program identified pictures containing horses, and is described in greater detail in [**?**]. Tests used 100 images containing horses, and 1086 control images with widely varying content; for a typical configuration, the program marks 11 images of horses and 4 control images.

24.4 Video

While video represents a richer source of information than still images, the issues remain largely the same. Videos are typically segmented into *shots* — short sequences that contain similar content — and techniques of the form described applied within shots. Because a large change between frames is a strong cue that a shot boundary has been reached, segmenting video into shots is usually done using measures of similarity like those described in section 24.2 (e.g. [**?**]).

The motion of individual pixels in a video is often called *optic flow* and is measured by attempting to find pixels in the next frame that correspond to a pixel in this (correspondence being measured by similarity in colour, intensity and texture). In principle, there is an optic flow vector at each pixel, forming a *motion field.* In practice, it is extremely hard to measure optic flow reliably at featureless pixels, because they could correspond to pretty much anything. For example, consider the optic flow of an egg rotating on its axis; there is very little information about what the pixels inside the boundary of the egg are doing, because each looks like the other.

Motion fields can be extremely complex; however, particularly if there are no moving objects in the frame, it is possible to classify motion fields corresponding to the camera shot used. For example, a pan shot will lead to strong lateral motion, and a zoom leads to a radial motion field. This classification is usually obtained by comparing the measured motion field with a parametric family (e.g. [**?**; **?**]).

Complex motion sequences are difficult to query without segmentation, because much of the motion may be irrelevant to the query; for example, in a soccer match, the motion of many players may not be significant. In Chang's system VideoQ, developed at Columbia, motion sequences are segmented moving blobs and then queried on the colour and motion of a particular blob (figure 24.17 and [**?**; **?**]).

The Informedia project at CMU has studied preparing detailed skims of video sequences. In this case, a segment of video is broken into shots, shots are annotated with the camera motion in shot, with the presence of faces, with the presence of

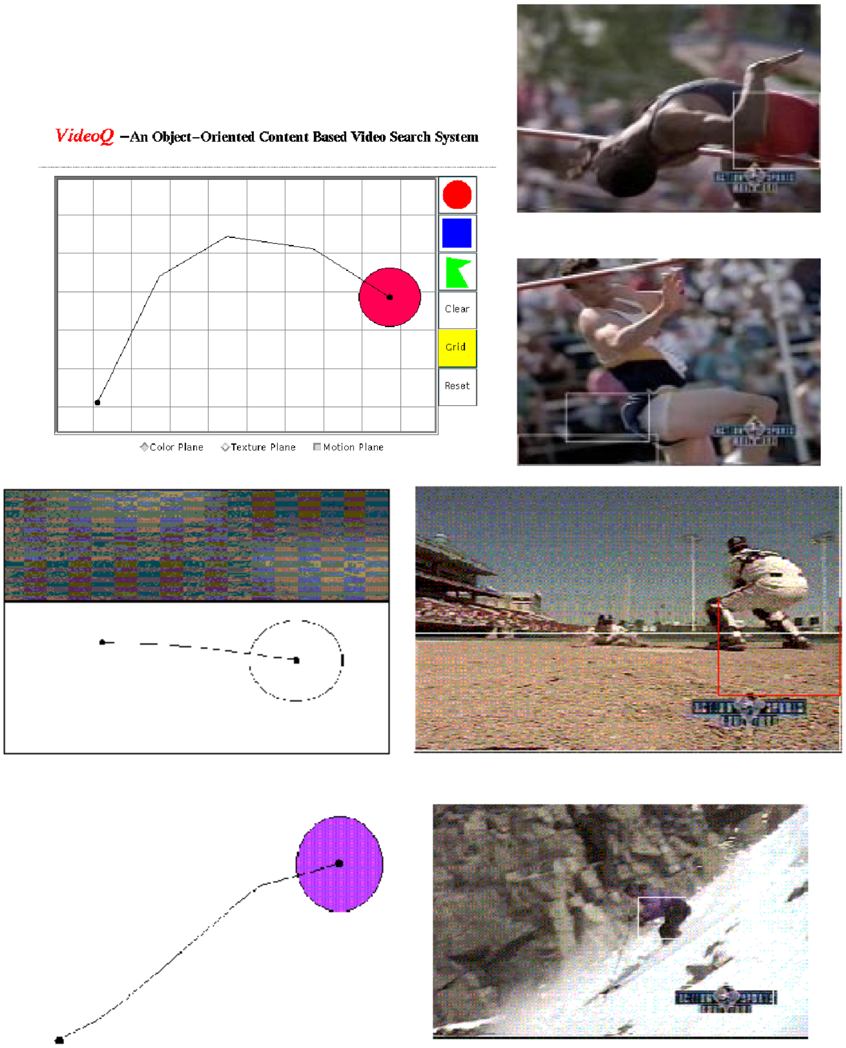

VideoQ –An Object–Oriented Content Based Video Search System
Clear
Grid
Reset
◇Color Plane
◇Texture Plane
☐ Motion Plane

Figure 24.17. *Video can be represented by moving blobs; sequences can then be queried by specifying blob properties and motion properties desired. The top left shows a query for a blob moving along a parabolic arc, sketched in the user interface for the VideoQ system. Top right shows frames from two sequences returned. As the center (baseball) and bottom (skiing) figures show, the mechanism extends to a range of types of motion. More details appear in [*?*]; figure by kind permission of S-F. Chang.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

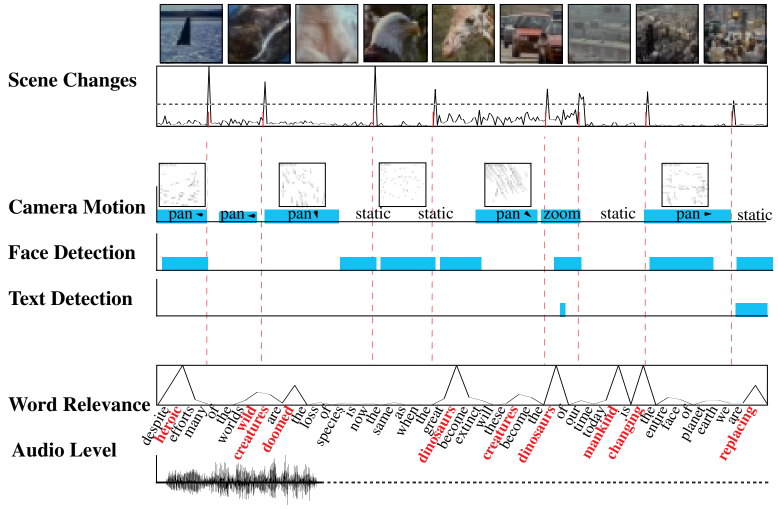


Figure 24.18. *Characterizing a video sequence to create a skim. The video is segmented into scenes. Camera motions are detected along with significant objects (faces and text). Bars indicate frames with positive results. Word relevance is evaluated in the transcript. More information appears in [**?**]; figure by kind permission of T. Kanade.* Permission granted for Library Trends article; reproduced in the fervent hope that permission will be granted for the book, too

text in shot, with keywords from the transcript and with audio level (figure 24.18). This information yields a compact representation — the “skim” — which gives the main content of the video sequence (details in [**?**; **?**] and [**?**; **?**].

24.5 Discussion

For applications where the colours, textures and layout of the image are all strongly correlated with the kind of content desired, a number of usable tools exist to find images based on content. There has been a substantial amount of work on user interfaces and browsing, although this work is usually done to get a system up and running, rather than through an explicit study of user practices. Because colour, texture and layout are at best a rough guide to image content, puzzling search results are pretty much guaranteed. There is not yet a clear theory of how to build interfaces that minimize the impact of this effect. The most widely adopted strategy is to allow quite fluid browsing.

When queries occur at a more semantic level, we encounter deep and poorly understood problems in object recognition. Object recognition seems to require segmenting images into coherent pieces and reasoning about the relationships between those pieces. This rather vague view of recognition can be exploited to produce segmented representations that allow searches for objects independent of their backgrounds; furthermore, some special cases of object recognition can be handled explicitly. It is not known how to build a system that could search for a wide variety of objects; building a user interface for such a system would present substantial problems, too. A natural research focus is the activities of people. In the next few years, it may be possible to find pictures of, say, a politician kissing a baby using entirely automatic methods.

Chapter 25

APPLICATION: IMAGE-BASED RENDERING

Why spend so much energy trying to reconstruct the three-dimensional structure of a scene from two, three or many pictures? This question may seem silly: after all, people do it all the time, using stereo, motion, and other depth cues, so it must be useful, if only to avoid bumping into things and getting hurt, or to distinguish interesting –and potentially dangerous– objects from the inoffensive background. Still, for many years, building a range map from a stereo pair or a motion sequence seemed more like an end than a means, and visual robot navigation remains, even today, a relatively small (but of course quite important) field of computer vision. Likewise, it has never been clearly established that stereopsis, or, for that matter, any kind of range data, provides a more suitable input to the recognition process than regular pictures (close one eye and look at a photograph of your mother, it is a safe bet that you will still recognize her; similar arguments hold for traditional application fields of stereo vision, say military intelligence).

But things are changing: the entertainment industry, from computer games to television advertising and feature films, demands synthetic pictures of real scenes, possibly mixed with images of artificial objects and actual film footage. This is what *image-based rendering* –defined here as the synthesis of new views of a scene from pre-recorded pictures– is all about, and the topic of this chapter. We present below a number of representative approaches to image-based rendering, dividing them, rather arbitrarily, into (1) techniques that first recover a three-dimensional scene model from a sequence of pictures, then render it with classical computer graphics tools (naturally, these approaches are often related to stereo and motion analysis, see Section 25.1), (2) methods that do not attempt to recover the camera or scene parameters, but construct instead an explicit representation of the set of all possible pictures of the observed scene, then use the image position of a small number of tie points to specify a new view of the scene and *transfer* all the other points into the new image, in the photogrammetric sense already mentioned in Chapter 11 (Section 25.2), and finally (3) approaches that model images by a two-dimensional set of light rays (or more precisely by the value of the radiance along these rays) and the set of all pictures of a scene by a four-dimensional set of rays, the *light field* (Section 25.3). Figure 25.1 illustrates this taxonomy.

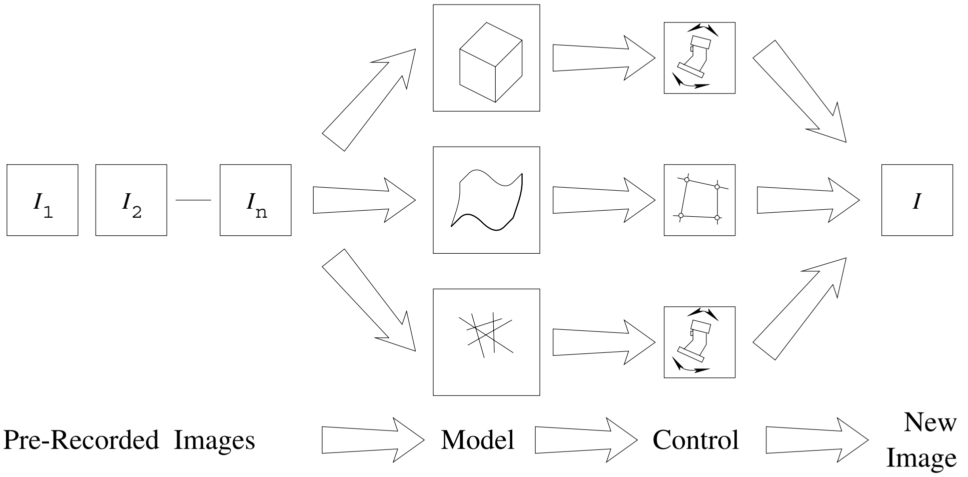


Figure 25.1. Approaches to image-based rendering. From top to bottom: three-dimensional model construction from image sequences; transfer-based image synthesis; the light field. From left to right, the diagram illustrates the image-based rendering pipeline: a model of the scene of interest (that may or may not be a three-dimensional model in the conventional sense of the world) is constructed from sample images and used to render new images of the scene. The rendering engine may be controlled by a joystick (or equivalently by the specification of rotation, translation and scale parameters), or in the case of transfer-based techniques, by setting the image position of a small number of tie points.

25.1 Constructing 3D Models from Image Sequences

This section addresses the problem of building and rendering a three-dimensional object model from a sequence of pictures. It is of course possible to construct such a model by fusing registered depth maps acquired by range scanners as described in Chapter 23, but we will focus instead in the rest of this section on the somewhat different case where the input images are digitized photographs or film clips of a rigid or dynamic scene.

25.1.1 Scene Modeling from Registered Images

We first examine the (relatively) simple case where a number or pictures of the same scene (say half a dozen, or more) have been taken under carefully controlled laboratory conditions, so that all images are registered in the same global coordinate system. We will address the more general case of cameras with unknown extrinsic parameters in the following sections.

Volumetric Reconstruction

Imagine that someone (maybe yourself, armed with your favorite mouse, or, why not, the elusive perfect segmentation program) has delineated the outline of an object in a collection of registered pictures. It is impossible to uniquely recover the object shape from these image contours, even from an arbitrary large (or for that matter, infinite) number of pictures since, as observed in Chapter 4, the concave portions of the object will be forever hidden from the keenest observer. Still, it seems that we should be able to reconstruct a reasonable approximation of the surface from a large enough (or maybe "interesting enough") set of pictures. There are two main global constraints imposed on a solid shape by its image contours: (1) the shape should lie in the volume defined by the intersection of the viewing cones attached to each image, and (2) the cones should be tangent to its surface (there are of course other local constraints: for example, as shown in Chapter 4, convex (resp. concave) parts of the contour should be the projections of convex (resp. saddle-shaped) parts of the surface). Baumgart exploited the first of these constraints in his 1974 PhD thesis as he constructed polyhedral models of various objects by intersecting the polyhedral cones associated with polygonal approximations of their silhouettes (Figure 25.2(a)). Interestingly, his goal was to construct object models appropriate for recognition rather than computer graphics (which was of course in its infancy then). It is probably fair to say that Baumgart's research did not really change the way we think about recognition. But it certainly had a great impact on solid modeling: Baumgart invented the so-called *Euler operators* that provide a robust approach to the computation of various boolean operations between solids. Closer to us, Baumgart's ideas spawned a number of approaches to object modeling from silhouettes, including the technique presented in the rest of this section [**?**], that also incorporates the tangency constraint associated with the viewing cones: as in Baumgart's system, a polyhedral approximation of the observed object is first constructed by intersecting the visual cones associated with a few registered photographs (Figure 25.2(b)). The vertices of this polyhedron are used as the control points of a smooth *spline surface*, which is deformed until it is tangent to the visual rays. We will focus on the second step of this approach, concerned with the construction and deformation of the spline surface.

Spline Construction. A *spline curve* is a piecewise-polynomial parametric curve that satisfies certain smoothness conditions [**?**]. For example, it may be C^k, i.e., differentiable with continuous derivatives of order up to k, with k usually taken to be 1 or 2, or G^k, i.e., not necessarily differentiable everywhere, but with continuous tangents in the G^1 case, and continuous curvatures in the G^2 case. Spline curves are usually constructed by patching together *Bézier arcs.* A Bézier curve of degree n is a polynomial parametric curve defined as the barycentric combination of $n+1$ *control points* P_i $(i = 0, 1, \ldots, n)$, i.e.,

$$P(t) = \sum_{i=0}^{n} b_i^{(n)}(t) P_i,$$

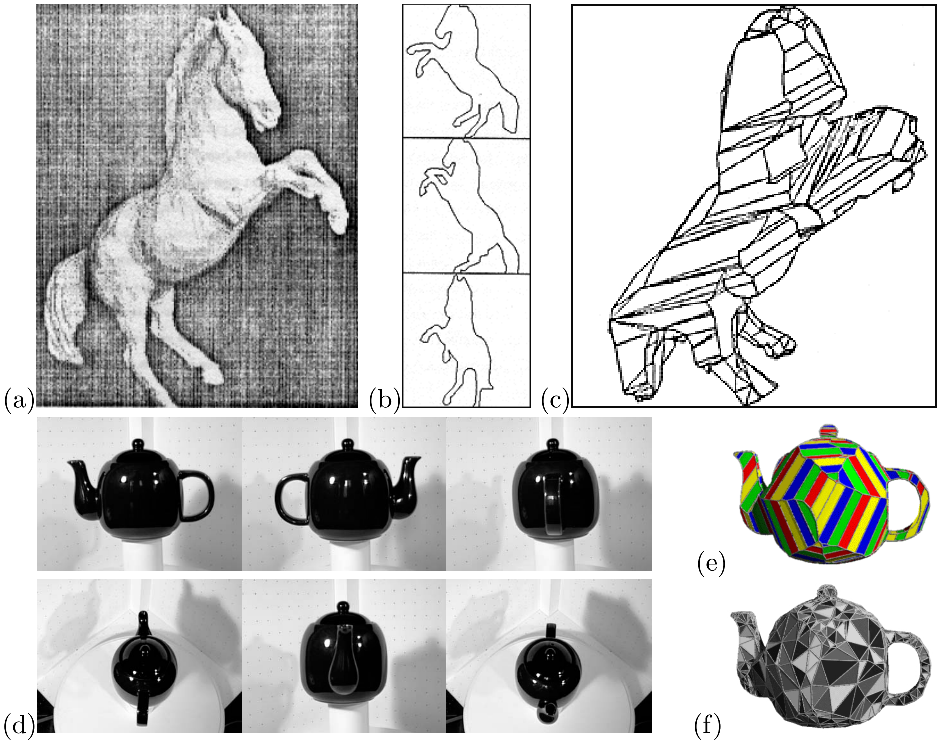


Figure 25.2. Constructing object models by intersecting (polyhedral) viewing cones. Yesterday: (a) sample photograph; (b) silhouettes; (c) polyhedral model. Reprinted from [**?**], Figure 9.6. Today: (d) six photographs of a teapot; (e) the raw intersection of the cones; (f) the triangulation obtained by splitting each face into triangles and simplifying the resulting mesh. Reprinted from [**?**], Figure 3.

where the polynomials $b_i^{(n)}(t) \overset{\text{def}}{=} \binom{n}{i} t^i (1-t)^{n-i}$ are called the *Bernstein polynomials* of degree n.[1] A Bézier curve interpolates its first and last control points, but not the other ones (Figure 25.3)(a). The tangents at its endpoints are along the first and last line segments of the *control polygon* formed by the control points.

The definition of Bézier arcs and spline curves naturally extends to surfaces: a triangular Bézier patch of degree n is a parametric surface defined as a barycentric combination of a triangular array of control points P_{ijk}:

$$P(u,v) = \sum_{i+j+k=n} b_{ijk}^n(u,v,1-u-v)P_{ijk},$$

---

[1] This is indeed a barycentric combination (as defined in Chapter 13) since the Bernstein polynomials are easily shown to always add to 1. In particular, Bézier curves are affine constructs, a very desirable property since it allows the definition of these curves purely in terms of their control points and independently of the choice of any external coordinate system.

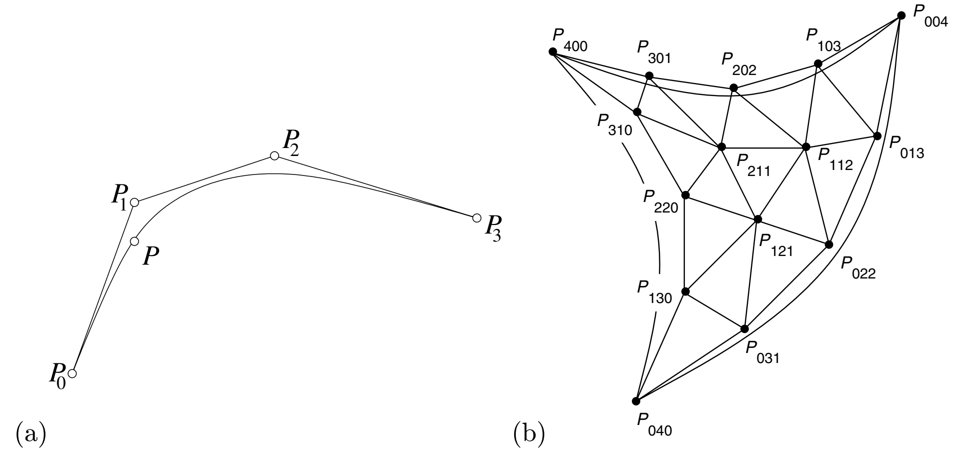


Figure 25.3. Bézier curves and patches: (a) a cubic Bézier curves and its control polygon; (b) a quartic triangular Bézier patch and its control mesh. *Tensor-product Bézier patches* can also be defined using a rectangular array of control points [**?**], and they are, in fact, more commonly used in computer-aided geometric design than their triangular counterparts. Triangular patches are, however, more appropriate for modeling free-form closed surfaces.

where the homogeneous polynomials $b^n_{ijk}(u, v, w) \stackrel{\text{def}}{=} \frac{n!}{i!j!k!} u^i v^j w^k$ are the *trivariate Bernstein polynomials* of degree n. In the rest of this section, we will use *quartic* ($n = 4$) Bézier patches, each defined by fifteen control points (Figure 25.3(b)).

By definition, a G^1 *triangular spline* is a network of triangular Bézier patches that share the same tangent plane along their common boundaries. A necessary (but not sufficient) condition for G^1 continuity is that all control points surrounding a common vertex be coplanar. These points define the boundary curves of the patch (each of which is itself a Bézier curve). We will first construct these points, then construct the remaining (internal) control points to ensure that the resulting spline is indeed G^1 continuous. As discussed in [**?**], a set of coplanar points Q_i $(i = 1, \dots, p)$ can be created as a barycentric combination of p other points C_j $(j = 1, \dots, p)$ in general position (in our case, the centroids of the p triangles T_j adjacent to a vertex V of the input triangulation, Figure 25.4(left)) as

$$Q_i = \sum_{j=1}^{p} \frac{1}{p} \left\{ 1 + \cos\frac{\pi}{p} \cos\left([2(j-i) - 1] \frac{\pi}{p} \right) \right\} C_j.$$

This construction places the points Q_i in a plane that passes through the centroid O of the points C_i. Translating this plane so that O coincides with the vertex V yields a new set of points A_i that all lie in a plane passing through V (Figure 25.4(center)).

Since cubic Bézier curves are defined by four points, we can interpret two adjacent vertices V and V' and the points A_i and A'_i associated with the corresponding

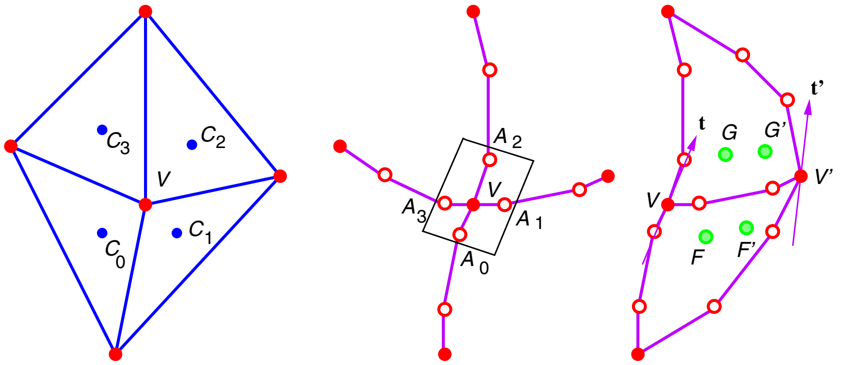


Figure 25.4. Construction of a triangular spline over a triangular polyhedral mesh. From left to right: the cubic boundary control points, the boundary curves surrounding a mesh vertex, and the construction of internal control points from tangent specification. After [**?**], Figures 1 and 2.

edge as the control points of a cubic curve. This construction yields a set of cubic curves that interpolate the vertices of the control mesh and form the boundaries of triangular patches. Once these curves have been constructed, the control points on both sides of a boundary can be chosen to satisfy inter-patch G^1 continuity. In this construction, the cross-boundary tangent field linearly interpolates the tangents at the two endpoints of the boundary curve. At the endpoint V, the tangent $\boldsymbol{t}$ across the curve that contains the point A_i is taken to be parallel to the line joining A_{i-1} to A_{i+1}. The tangent $\boldsymbol{t}'$ is obtained by a similar construction. The interior control points F, F' and G, G' (Figure 25.4(right)) are constructed by solving the set of linear equations obtained associated with this geometric condition [**?**]. However, there are not enough degrees of freedom in a quartic patch to allow the simultaneous setting of the interior points for all three boundaries. Thus each patch must be split three ways, using for example the method of Shirman and Sequin [**?**] to ensure continuity among the new patches: performing *degree elevation* on the boundary curves replaces them by quartic Bézier curves with the same shape (see exercises). Three quartic triangular patches can then be constructed from the boundaries as shown in Figure 25.5. The result is a set of three quartic patches for each mesh face which are G^1 continuous across all boundaries.

Spline Deformation. Given a triangulation such as the one shown in Figure 25.2(b) it is possible to construct a G^1 triangular spline approximation of the modeled object's surface. This section shows how to deform this spline to ensure that it is tangent to the viewing cones associated with the input photographs. The shape of the spline surface S is determined by the position of its control vertices V_j $(j = 1, \ldots, p)$. We denote by V_{jk} $(k = 1, 2, 3)$ the coordinates of the point V_j in some reference Euclidean coordinate system, and use these $3p$ coefficients as shape parameters.

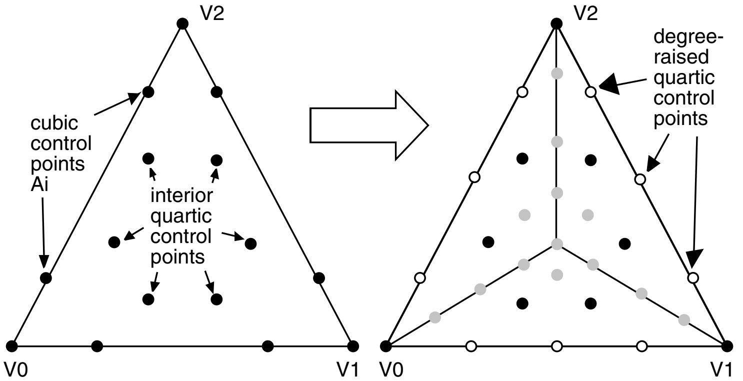

V2
cubic control points Ai
interior quartic control points
V0
V1
V2
degree-raised quartic control points
V0
V1

Figure 25.5. Splitting a patch three ways to enforce G^1 continuity: the white points are the control points obtained by raising the degree of the control curves, and the grey points are the remaining control points, computed to ensure G^1 continuity. Reprinted from [**?**], Figure 2.

Given a set of rays R_i $(i = 1, \dots, q)$, we minimize the energy function

$$\frac{1}{q}\sum_{i=1}^{q} d^2(R_i, S) + \lambda \sum_{i=1}^{r} \iint [|P_{uu}|^2 + 2|P_{uv}|^2 + |P_{vv}|^2]\, dudv$$

with respect to the parameters V_{jk} of S. Here, $d(R, S)$ denotes the distance between the ray R and the surface S, the integral is a *thin-plate* spline energy term used to enforce smoothness in areas of sparse data, and λ is a constant weight introduced to balance the distance and smoothness terms. The variables u and v in this integral are the patch parameters and the summation is done over the r patches that form the spline surface. The signed distance between a ray and a surface patch can be computed using Newton's method. For rays that do not intersect the surface, we define $d(R, S) = \min\{|\overrightarrow{QP}|, Q \in R, P \in S\}$, and compute the distance by minimizing $|\overrightarrow{QP}|^2$. For those rays which intersect the surface, we follow Brunie, Lavallée, and Szeliski [**?**] and measure the distance to the *farthest* point from the ray that lies on the surface in the direction of the surface normal at the corresponding occluding contour point. In both cases, Newton iterations are initialized from a sampling of the surface S.

During surface fitting, we deform the spline to minimize the mean-squared ray-surface distance using a simple gradient descent technique. Although each distance is computed numerically, its derivatives with respect to the surface parameters V_{jk} are easily computed by differentiating the constraints satisfied by the surface and ray points where the distance is reached (see exercises).

This three object models shown in Figure 25.6 have been constructed with the method described in this section. This technique does not require establishing any correspondence across the input pictures, but the scope of its current implementation is limited to static scenes. In contrast, the approach presented next is based

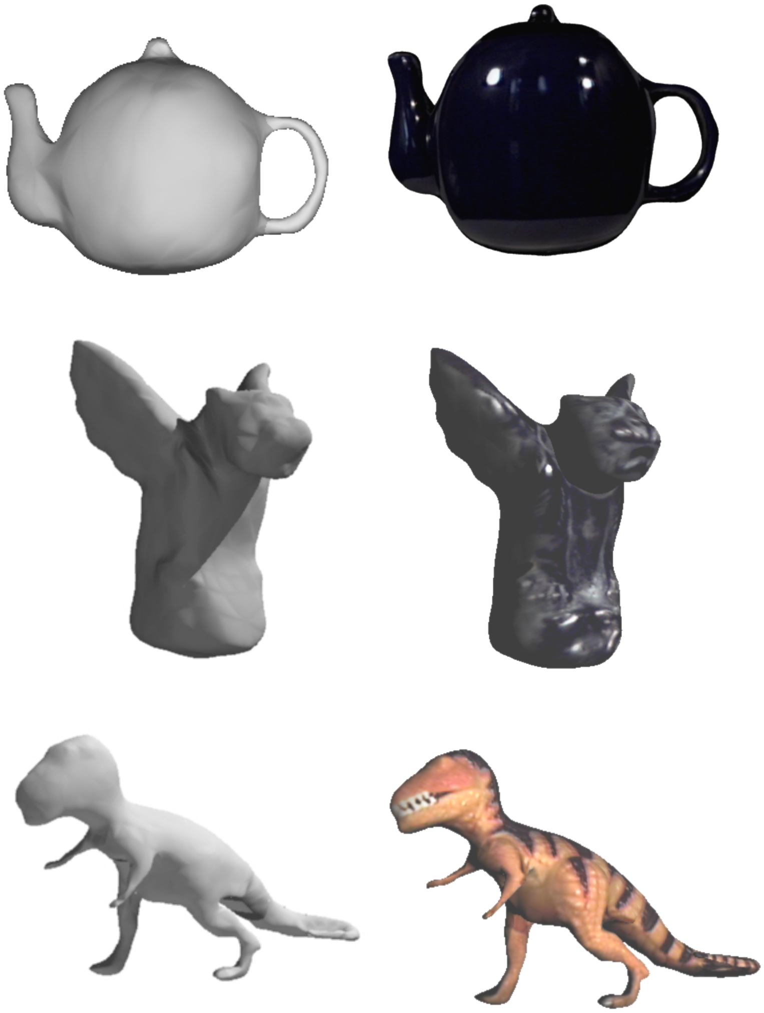


Figure 25.6. Shaded and texture-mapped models of a teapot, a gargoyle and a dinosaur. The teapot was constructed from six registered photographs; the gargoyle and the dinosaur models were each built from nine images. Reprinted from [**?**], Figures 4 and 5.

on multi-camera stereopsis, and as such requires correspondences, but it handles dynamic scenes as well as static ones.

Virtualized Reality

Kanade and his colleagues [**?**] have proposed the concept of *Virtualized Reality* as a new visual medium for manipulating and rendering pre-recorded and synthetic images of real scenes captured in a controlled environment. The first physical implementation of this concept at Carnegie-Mellon University consisted of a geodesic dome equipped with 10 synchronized video cameras hooked to consumer-grade VCRs. As of this writing, the latest implementation is a "3D Room" (Figure 25.7), where a volume of $20 \times 20 \times 9$ cubic feet is observed by 49 color cameras connected to a PC cluster, with the capability of digitizing in real-time the synchronized video streams of all cameras.

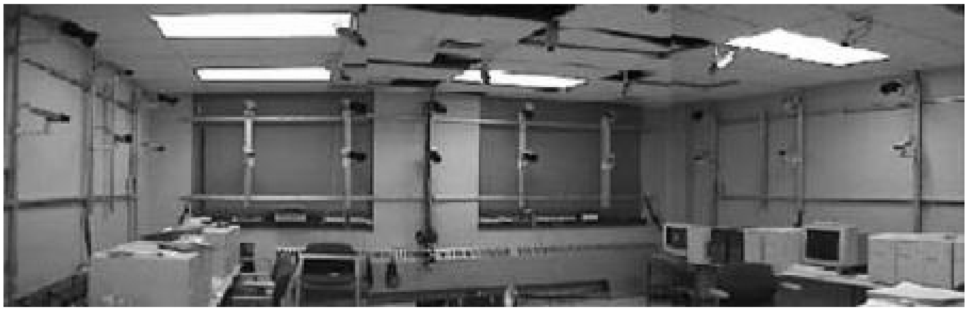


Figure 25.7. The 3D Room. There are 49 cameras total, ten of which are mounted on each wall, with the remaining nine cameras mounted on the ceiling. Reprinted from [**?**], Figure 1.

The internal and external parameters of all cameras are first measured in the same world coordinate system using the algorithm proposed by Tsai [**?**] and presented in Chapter 5. Three-dimensional scene models are then acquired by fusing dense depth maps acquired via multi-baseline stereo (see [**?**] and Chapter 12). One such map is acquired by each camera and a small number of its neighbors (between three and six). Every range image is then converted to a surface mesh that can be rendered using classical computer graphics techniques such as texture mapping. As shown by Figure 25.8, images of a scene constructed from a single depth map may exhibit gaps. These gaps can be filled by rendering in the same image the meshes corresponding to adjacent cameras.

It is also possible to directly merge the surface meshes associated with different cameras into a single surface model. This task is challenging since: (1) multiple, conflicting measurements of the same surface patches are available in areas where the fields of view of several cameras overlap, and (2) certain scene patches are not observed by any camera. Both problems can be solved using the volumetric technique for range image fusion proposed by Curless and Levoy [**?**] and introduced in Chapter 23.

Once a global surface model has been constructed, it can of course be texture-mapped as before. Synthetic animations can also be obtained by interpolating two

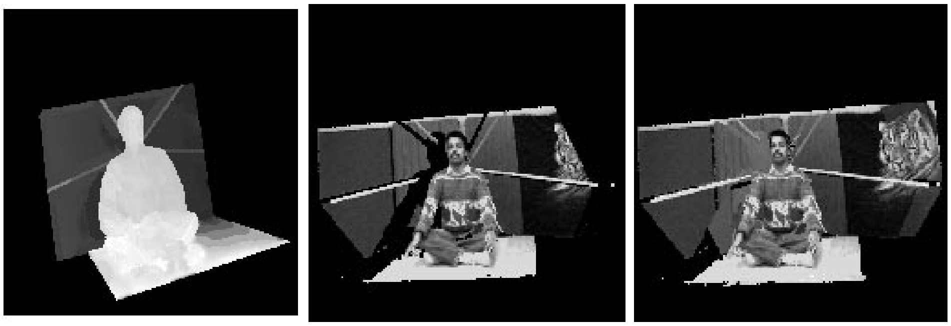


Figure 25.8. Multi-baseline stereo. From left to right: the range map associated with a cluster of cameras; a texture-mapped image of the corresponding mesh, observed from a different viewpoint; note the dark areas associated with depth discontinuities in the map; a texture-mapped image constructed from two adjacent camera clusters; note that the gaps have been filled. Reprinted from [**?**], Figures 3 and 8.

arbitrary views in the input sequence. First, the surface model is used to establish correspondences between these two views: the optical ray passing through any point in the first image is intersected with the mesh and the intersection point is reprojected in the second image, yielding the desired match.[2] Once the correspondences are known, new views are constructed by linearly interpolating both the positions and colors of matching points. As discussed in [**?**], this simple algorithm only provides an approximation of true perspective imaging, and additional logic has to be added in practice to handle points that are visible in the first image but not in the second one. Nevertheless, it can be used to generate realistic animations of dynamic scenes with changing occlusion patterns, as demonstrated by Figure 25.9.

25.1.2 Scene Modeling from Unregistered Images

This section addresses again the problem of acquiring and rendering three-dimensional object models from a set of images, but, this time, the positions of the cameras observing the scene are not known a priori and they must be recovered from image information using methods related to those presented in Chapters 13 and 14. The techniques presented in this section are, however, explicitly geared toward computer graphics applications.

The Façade System

The *Façade* system for modeling and rendering architectural scenes from digitized photographs was developed at UC Berkeley by Debevec, Taylor and Malik [**?**]. This

---

[2] Classical narrow-baseline methods like correlation would be ineffective in this context since the two views may be very far from each other. A similar method is used in the Façade system described later in this chapter to establish correspondences between widely separated images when the rough shape of the observed surface is known.

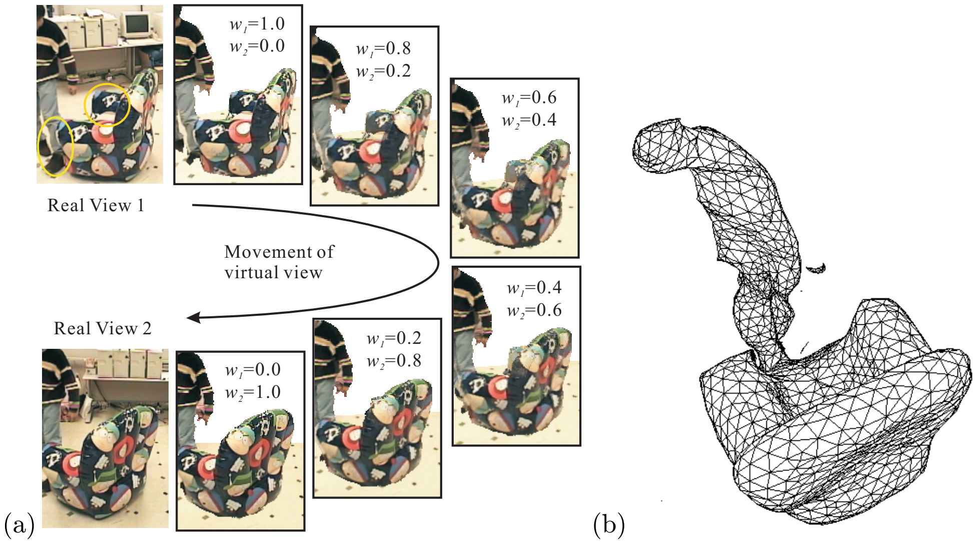

w_1=1.0
w_2=0.0
w_1=0.8
w_2=0.2
w_1=0.6
w_2=0.4
Real View 1
Movement of virtual view
Real View 2
w_1=0.4
w_2=0.6
w_1=0.2
w_2=0.8
w_1=0.0
w_2=1.0
(a)
(b)

Figure 25.9. Virtualized Reality: (a) a sequence of synthetic images; note that occlusion in the two elliptical regions of the first view is handled correctly; (b) the corresponding mesh model. Reprinted from [**?**], Figures 5 and 11.

system takes advantage of the relatively simple overall geometry of many buildings to simplify the estimation of scene structure and camera motion, and it uses the simple but powerful idea of *model-based stereopsis*, to be described in a minute, to add detail to rough building outlines. Figure 25.10 shows an example.

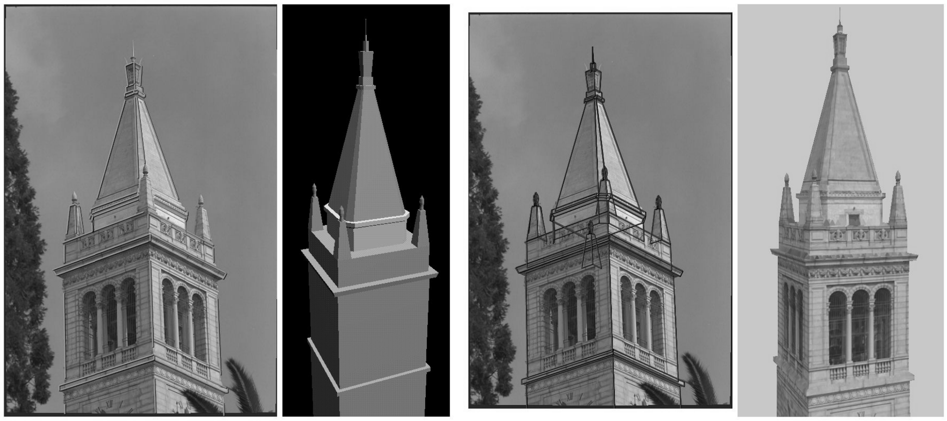


Figure 25.10. Façade model of the Berkeley Campanile. From left to right: a photograph of the Campanile, with selected edges overlaid; the 3D model recovered by photogrammetric modeling; reprojection of the models into the photograph; a texture-mapped view of the model. Reprinted from [**?**], Figure 2.

Façade models are constrained hierarchies of parametric primitives such as boxes, prisms and solids of revolution. These primitives are defined by a small number of coefficients (e.g., the height, width, and breadth of a box) and related to each other by rigid transformations. Any of the parameters defining a model is either a constant or a variable, and constraints can be specified between the various unknowns (e.g., two blocks may be constrained to have the same height). Model hierarchies are defined interactively with a graphical user interface, and the main computational task of the Façade system is to use image information to assign definite values to the unknown model parameters.

The overall system is divided into three main components: The first one, or *photogrammetric module*, recasts structure and motion estimation as a non-linear optimization problem involving relatively few variables, namely the positions and orientations of the cameras used to photograph a building and the parameters of the building model. The input to this optimization procedure is a set of correspondences between line segments selected by hand in the photographs and the corresponding parts of the parametric model. We saw in Chapter 5 that the mapping between a line with Plücker coordinate vector $\boldsymbol{\Delta}$ and its image with homogeneous coordinates $\boldsymbol{\delta}$ can be represented by $\rho\boldsymbol{\delta} = \tilde{\mathcal{M}}\boldsymbol{\Delta}$, where $\tilde{\mathcal{M}}$ is a 3×6 matrix whose row vectors are exterior products of the rows of the projection matrix $\mathcal{M}$. Here, $\boldsymbol{\Delta}$ is a function of the model parameters, and $\tilde{\mathcal{M}}$ depends on the corresponding camera position and orientation.

Let us consider an image edge e of length l, whose endpoints have homogeneous coordinates $\boldsymbol{p}_0 = (u_0, v_0, 1)^T$ and $\boldsymbol{p}_1 = (u_1, v_1, 1)^T$. To measure the discrepancy between e and the predicted line $\boldsymbol{\delta}$, it is convenient to represent the points of e as barycentric combinations of $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ and introduce the signed distance $h : [0,1] \rightarrow \mathbb{R}$ between the points of e and $\boldsymbol{\delta}$. Following Chapter 5, we have

$$h(t) = \frac{1}{|[\boldsymbol{\delta}]_2|}\boldsymbol{p}(t) \cdot \boldsymbol{\delta} = \frac{1}{|[\tilde{\mathcal{M}}\boldsymbol{\Delta}]_2|}[(1-t)\boldsymbol{p}_0 + t\boldsymbol{p}_1]^T \tilde{\mathcal{M}}\boldsymbol{\Delta},$$

where $[\boldsymbol{a}]_2$ denotes the vector formed by the first two coordinates of the vector $\boldsymbol{a} \in \mathbb{R}^3$. In particular, the discrepancy measure can be chosen to be the integral of h^2 over the edge, which, in turn, is easily shown to be equal to

$$E = \int_0^1 h^2(t)dt = \frac{l}{3}(h(0)^2 + h(0)h(1) + h(1)^2),$$

and the recovery of the model and camera parameters reduces to the non-linear minimization of the average of this measure taken over all edge correspondences and cameras. As shown in [**?**] and the exercises, when the orientation of some of the model edges is fixed relative to the world coordinate system, an initial estimate for these parameters is easily found using linear least squares.

The second main component of Façade is the *view-dependent texture-mapping module*, that renders an architectural scene by mapping different photographs onto its geometric model according to the user's viewpoint (Figure 25.11). Conceptually, the cameras are replaced by slide projectors that project the original images onto the model. Of course, each camera will only see a portion of a building (Figure 25.11(top)), and several photographs must be used to render a complete model. On the other hand, certain parts of a building will in general be observed by several cameras, so the renderer must not only pick, but also appropriately merge, the pictures relevant to the synthesis of a virtual view. The solution adopted in Façade is to assign to each pixel in a new image a weighted average of the values predicted from the overlapping input pictures, with weights inversely proportional to the angle between the corresponding light rays in the input and virtual views (Figure 25.11(bottom)).

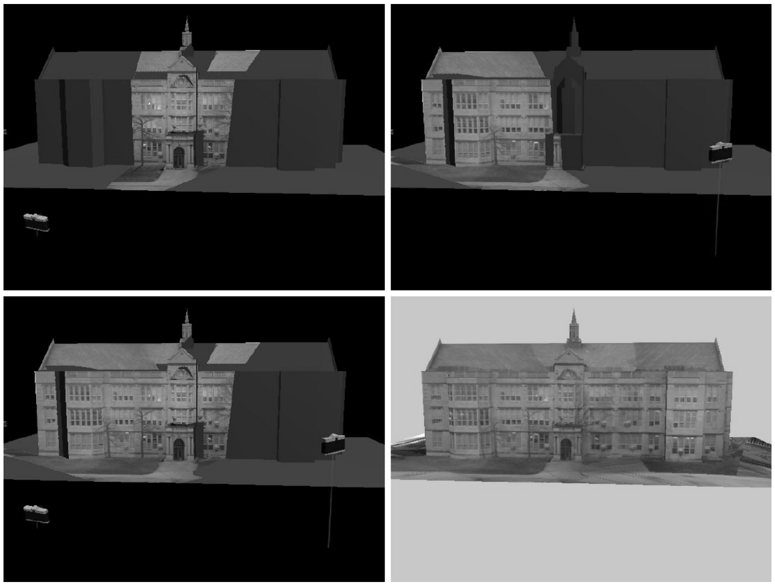


Figure 25.11. View-dependent texture mapping: the top part of the figure shows images projected onto a model from two camera positions (note how portions of the model that lie within the viewing frustrum of each camera are shadowed by the model itself and thus cannot be texture-mapped). The bottom-left part of the figure is a composite of these two pictures obtained with the weighted-average method described in the text. The bottom-right image is a composite obtained from twelve photographs. Reprinted from [**?**], Figure 12.

The last component of Façade is the *model-based stereopsis module*, that uses stereo pairs to add fine geometric detail to the relatively rough scene description constructed by the photogrammetric modeling module. The main difficulty in using stereo vision in this setting is the wide separation of the cameras, which prevents the straightforward use of correlation-based matching techniques. The solution adopted in Façade is to exploit a priori shape information to map the stereo images into the same reference frame (Figure 25.12(top)). Specifically, given *key* and *offset* pictures, the offset image can be projected onto the scene model before being rendered from the key camera's viewpoint, yielding a *warped offset* picture very similar to the key image (Figure 25.12(bottom)). In turn, this allows the use of correlation to establish correspondences between these two images, and thus between the key and offset images as well. Once the matches between these two pictures have been established, stereo reconstruction reduces to the usual triangulation process.

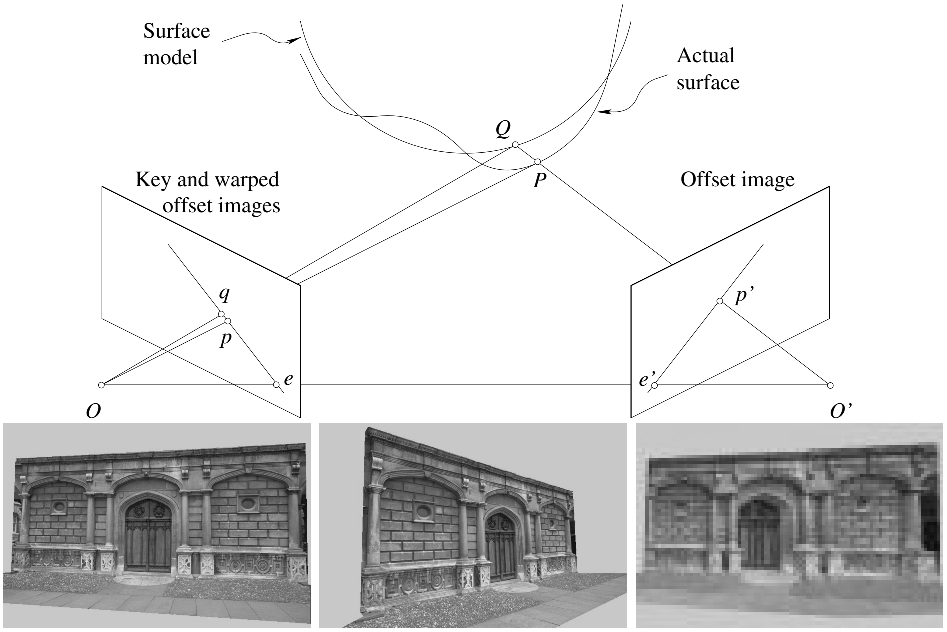


Figure 25.12. Model-based stereopsis. Top: synthesis of a warped offset image. The point p' in the offset image is mapped onto the point Q of the surface model, then reprojected onto the point q of the warped offset image. The actual surface point P observed by both cameras projects onto the point p of the key image. Note that the point q must lie on the epipolar line ep, which facilitates the search for matches as in the conventional stereo case. Note also that the disparity between p and q along the epipolar line measures the discrepancy between the modelled and actual surfaces. After [**?**], Figure 15. Bottom, from left to right: a key image, an offset image, and the corresponding warped offset image. Reprinted from [**?**], Figure 13.

25.2 Transfer-Based Approaches to Image-Based Rendering

This section explores a completely different approach to image-based rendering. In this framework, an explicit three-dimensional scene reconstruction is never performed. Instead, new images are created directly from a (possibly small) set of views among which point correspondences have been established by feature tracking or conventional stereo matching. This approach is related to the classical transfer problem from photogrammetry, already mentioned in Chapter 11: given the image positions of a number of tie points in a set of reference images and in a new image, and given the image positions of a ground point in the reference images, predict the position of that point in the new image.

Transfer-based techniques for image-based rendering were introduced in the projective setting by Laveau and Faugeras [**?**], who proposed to first estimate the pairwise epipolar geometry between reference views, then reproject the scene points into a virtual image, itself specified by the positions of the new optical center in two reference pictures (i.e., the epipoles) and the position of four tie points in the new view. By definition, the epipolar geometry constrains the possible reprojections of points in the reference images. In the new view, the projection of the scene point is at the intersection of two epipolar lines associated with the point and two reference pictures. Once the feature points have been reprojected, realistic pictures are synthesized using ray tracing and texture mapping.

As noted by Laveau and Faugeras, however, since the Euclidean constraints associated with calibrated cameras are not enforced, the rendered images are in general separated from the "correct" pictures by arbitrary planar projective transformations unless additional scene constraints are taken into account. The rest of this section explores two affine variants of the transfer-based approach that circumvent this difficulty. Both techniques construct a parameterization of the set of all images of a rigid scene: in the first case (Section 25.2.1), the vector space structure of the affine image space is used to render synthetic objects in an augmented reality system. Because the tie points in this case are always geometrically valid image features (e.g., the corners of calibration polygons, see Figure 25.13), the synthesized images are automatically Euclidean ones. In the second instance (Section 25.2.2), the metric constraints associated with calibrated cameras are explicitly taken into account in the image space parameterization, guaranteeing once again the synthesis of correct Euclidean images.

Let us note again a particularity of transfer-based approaches to image-based rendering, already mentioned in the introduction: because no three-dimensional model is ever constructed, a joystick cannot be used to control the synthesis of an animation: instead, the position of tie points must be specified interactively by a user. This is not a problem in an augmented reality context, but whether this is a viable user interface for virtual reality applications remains to be shown.

25.2.1 Affine View Synthesis

Here we address the problem of synthesizing new (affine) images of a scene from old ones, *without* setting an explicit three-dimensional Euclidean coordinate system. Recall from Chapter 13 that if we denote the coordinate vector of a scene point P in some world coordinate system by $\boldsymbol{P} = (x, y, z)^T$, and denote by $\boldsymbol{p} = (u, v)^T$ the coordinate vector of the projection p of P onto the image plane, the affine camera model can be written as

$$\boldsymbol{p} = \boldsymbol{o} + \mathcal{M}\boldsymbol{P}, \quad \text{where} \quad \mathcal{M} = \begin{pmatrix} \boldsymbol{a}^T \\ \boldsymbol{b}^T \end{pmatrix}, \tag{25.2.1}$$

$\boldsymbol{o}$ is the position of the projection into the image of the object coordinate system's origin, and $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors in $\mathbb{R}^3$.

Let us consider four (non-coplanar) scene points, say P_0, P_1, P_2 and P_3. We can choose (without loss of generality) these points as an affine world basis so their coordinate vectors are

$$\boldsymbol{P}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{P}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{P}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \boldsymbol{P}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Of course the points P_i $(i = 1, 2, 3)$ will *not* in general be at a unit distance from P_0, nor will the vectors $\overrightarrow{P_0P_i}$ and $\overrightarrow{P_0P_j}$ be orthogonal to each other when $i \neq j$. This is irrelevant since we work in an affine setting. Since the 3×3 matrix whose columns are $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ and $\boldsymbol{P}_3$ is the identity matrix, (25.2.1) can be rewritten as

$$\boldsymbol{p} = \boldsymbol{o} + \mathcal{M}\boldsymbol{P} = \boldsymbol{o} + \begin{pmatrix} \boldsymbol{a}^T \\ \boldsymbol{b}^T \end{pmatrix} [\boldsymbol{P}_1 | \boldsymbol{P}_2 | \boldsymbol{P}_3] \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Finally, since we have chosen P_0 as the origin of the world coordinate system, we have $\boldsymbol{o} = \boldsymbol{p}_0$ and we obtain

$$\boldsymbol{p} = \boldsymbol{p}_0 + x\boldsymbol{p}_1 + y\boldsymbol{p}_2 + z\boldsymbol{p}_3. \tag{25.2.2}$$

This is not terribly surprising (affine projections preserve affine coordinates), but interesting: for example, it follows from (25.2.2) that x, y and z can be computed from $m \geq 2$ images through linear least squares. Once these values are known, new images can be generated by specifying the image positions of the points p_0, p_1, p_2, p_3 and using (25.2.2) to compute all the other point positions. In addition, since the affine representation of the scene is truely three-dimensional, the relative depth of scene points can be computed and used to eliminate hidden surfaces in the z-buffer part of the graphics pipeline. This is the method proposed by Kutulakos and Vallino [**?**], and it is of course intimately related to the method proposed by Koenderink and Van Doorn [**?**] to estimate affine structure from two images.

It should be noted that specifying arbitrary positions for the points p_0, p_1, p_2, p_3 will (in general) give rise to affinely-deformed pictures. This is not a problem in augmented reality applications, where graphical and physical objects co-exist in the image. In this case, the anchor points p_0, p_1, p_2, p_3 can be chosen among true image points, guaranteed to be in the correct Euclidean position. Figure 25.13 shows an example, where synthetic objects have been overlaid on real images.

This approach can be extended to longer input image sequences. Suppose we observe a fixed set of points P_i $(i = 0, \ldots, n-1)$ with coordinate vectors $\boldsymbol{P}_i$, and let $\boldsymbol{p}_i$ denote the coordinate vectors of the corresponding image points. Writing

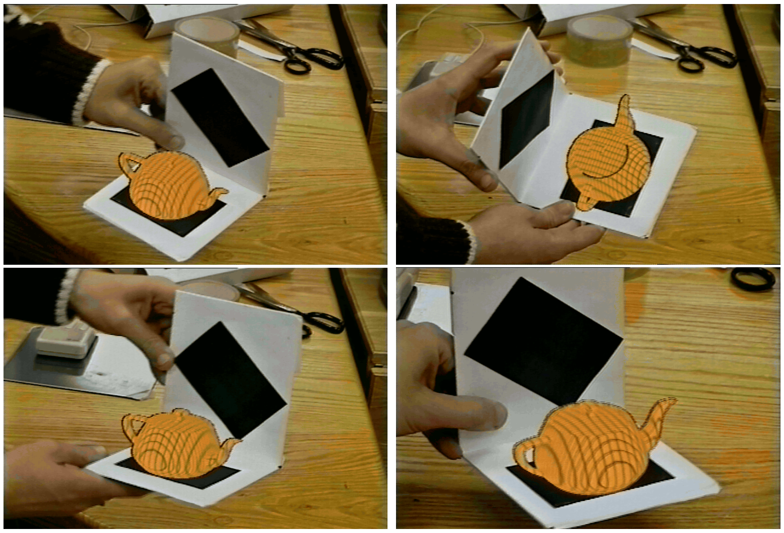


Figure 25.13. Augmented reality experiment. The (affine) world coordinate system is defined by corners of the black polygons. Reprinted from [?], Figure 14.

(25.2.1) for all the scene points yields

$$\begin{pmatrix} \boldsymbol{p}_0 \\ \dots \\ \boldsymbol{p}_{n-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{P}_0^T & \mathbf{0}^T & 1 & 0 \\ \mathbf{0}^T & \boldsymbol{P}_0^T & 0 & 1 \\ \dots & \dots & \dots & \dots \\ \boldsymbol{P}_{n-1}^T & \mathbf{0}^T & 1 & 0 \\ \mathbf{0}^T & \boldsymbol{P}_{n-1}^T & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{o} \end{pmatrix}.$$

In other words, the set of affine images of n points is an eight-dimensional vector space V embedded in $\mathbb{R}^{2n}$ and parameterized by the vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{o}$. Given $m \geq 2$ views of the n points, a basis for this vector space can be identified by performing the singular value decomposition of the $2n \times m$ matrix

$$\begin{pmatrix} \boldsymbol{p}_0^{(1)} & \dots & \boldsymbol{p}_0^{(m)} \\ \dots & \dots & \dots \\ \boldsymbol{p}_{n-1}^{(1)} & \dots & \boldsymbol{p}_{n-1}^{(m)} \end{pmatrix}$$

where $\boldsymbol{p}_i^{(j)}$ denotes the position of the image point number i in frame number j. Once a basis for V has been constructed, new images can be constructed by assigning arbitrary values to $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{o}$. For interactive image synthesis purposes, a more intuitive control of the imaging geometry can be obtained by specifying as before the position of four image points, solving for the corresponding values of $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{o}$, then computing the remaining image positions.

25.2.2 Euclidean View Synthesis

As discussed earlier, a drawback of the method presented in the previous section is that specifiying arbitrary positions for the points p_0, p_1, p_2, p_3 will (in general) yield affinely-deformed pictures. This can be avoided by taking into account from the start the Euclidean constraints associated with calibrated cameras: we saw in Chapter 13 that a weak perspective camera is an affine camera satisfying the two constraints

$$\boldsymbol{a} \cdot \boldsymbol{b} = 0 \quad \text{and} \quad |\boldsymbol{a}|^2 = |\boldsymbol{b}|^2. \tag{25.2.3}$$

The previous section showed that the affine images of a fixed scene form an eight-dimensional vector space V. Now, if we restrict our attention to weak perspective cameras, the set of images becomes the six-dimensional subspace defined by the two polynomial constraints (25.2.3). Similar constraints apply to paraperspective and true perspective projection, and they also define a six-dimensional variety (i.e., a subspace defined by polynomial equations) in each case (see [**?**] and the exercises).

Let us suppose that we observe three points P_0, P_1, P_2 whose images are not collinear. We can choose (without loss of generality) a Euclidean coordinate system such that the coordinate vectors of the four points in this system are

$$\boldsymbol{P}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{P}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{P}_0 = \begin{pmatrix} p \\ q \\ 0 \end{pmatrix},$$

where p and q are nonzero but (a priori) unknown. Let us denote as before by p_i the projection of the point P_i $(i = 0, 1, 2)$. Since P_0 is the origin of the world coordinate system, we have $\boldsymbol{o} = \boldsymbol{p}_0$. We are also free to pick p_0 as the origin of the image coordinate system (this amounts to submitting all image points to a known translation), so (25.2.1) simplifies into

$$\boldsymbol{p} = \mathcal{M}\boldsymbol{P} = \begin{pmatrix} \boldsymbol{a}^T\boldsymbol{P} \\ \boldsymbol{b}^T\boldsymbol{P} \end{pmatrix}. \tag{25.2.4}$$

Now, applying (25.2.4) to P_1, P_2 and P yields

$$\boldsymbol{u} \stackrel{\text{def}}{=} \begin{pmatrix} u_1 \\ u_2 \\ u \end{pmatrix} = \mathcal{A}\boldsymbol{a} \quad \text{and} \quad \boldsymbol{v} \stackrel{\text{def}}{=} \begin{pmatrix} v_1 \\ v_2 \\ v \end{pmatrix} = \mathcal{A}\boldsymbol{b}, \tag{25.2.5}$$

where

$$\mathcal{A} \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{P}_1^T \\ \boldsymbol{P}_2^T \\ \boldsymbol{P}^T \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ p & q & 0 \\ x & y & z \end{pmatrix}.$$

In turn, this implies that

$$\boldsymbol{a} = \mathcal{B}\boldsymbol{u} \quad \text{and} \quad \boldsymbol{b} = \mathcal{B}\boldsymbol{v}, \tag{25.2.6}$$

where

$$\mathcal{B} \overset{\text{def}}{=} \mathcal{A}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \lambda & \mu & 0 \\ \alpha/z & \beta/z & 1/z \end{pmatrix} \quad \text{and} \quad \begin{cases} \lambda \overset{\text{def}}{=} -p/q, \\ \mu \overset{\text{def}}{=} 1/q, \\ \alpha \overset{\text{def}}{=} -(x+\lambda y) \\ \beta \overset{\text{def}}{=} -\mu y. \end{cases}$$

Using (25.2.6) and letting $\mathcal{C} \overset{\text{def}}{=} z^2\mathcal{B}^T\mathcal{B}$, the weak perspective constraints (13.4.2) can be rewritten as

$$\begin{cases} \boldsymbol{u}^T\mathcal{C}\boldsymbol{u} - \boldsymbol{v}^T\mathcal{C}\boldsymbol{v} = 0, \\ \boldsymbol{u}^T\mathcal{C}\boldsymbol{v} = 0, \end{cases} \tag{25.2.7}$$

with

$$\mathcal{C} = \begin{pmatrix} \xi_1 & \xi_2 & \alpha \\ \xi_2 & \xi_3 & \beta \\ \alpha & \beta & 1 \end{pmatrix} \quad \text{and} \quad \begin{cases} \xi_1 = (1+\lambda^2)z^2 + \alpha^2, \\ \xi_2 = \lambda\mu z^2 + \alpha\beta, \\ \xi_3 = \mu^2 z^2 + \beta^2. \end{cases}$$

Equation (25.2.7) defines a pair of linear constraints on the coefficients ξ_i ($i = 1,2,3$), α and β; they can be rewritten as

$$\begin{pmatrix} \boldsymbol{d}_1^T \\ \boldsymbol{d}_2^T \end{pmatrix} \boldsymbol{\xi} = 0, \tag{25.2.8}$$

where

$$\boldsymbol{d}_1 \overset{\text{def}}{=} \begin{pmatrix} u_1^2 - v_1^2 \\ 2(u_1u_2 - v_1v_2) \\ u_2^2 - v_2^2 \\ 2(u_1u - v_1v) \\ 2(u_2u - v_2v) \\ u^2 - v^2 \end{pmatrix}, \quad \boldsymbol{d}_2 \overset{\text{def}}{=} \begin{pmatrix} u_1v_1 \\ u_1v_2 + u_2v_1 \\ u_2v_2 \\ u_1v + uv_1 \\ u_2v + uv_2 \\ uv \end{pmatrix} \quad \text{and} \quad \boldsymbol{\xi} \overset{\text{def}}{=} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \alpha \\ \beta \\ 1 \end{pmatrix}.$$

When the four points P_0, P_1, P_2, and P are rigidly attached to each other, the five structure coefficients ξ_1, ξ_2, ξ_3, α and β are fixed. For a rigid scene formed by n points, choosing three of the points as a reference triangle and writing (25.2.8) for the remaining ones yields a set of $2n-6$ quadratic equations in $2n$ unknowns, which do indeed define a parameterization of the set of all weak perspective images of the scenes. This is the *Parameterized Image Variety* (or *PIV*) of Genc and Ponce [**?**].

Note again that the weak perspective constraints (25.2.8) are linear in the five structure coefficients. Thus, given a collection of images and point correspondences, these coefficients can be estimated through linear least squares. Once the vector $\boldsymbol{\xi}$ has been estimated, arbitrary image positions can be assigned to the three reference points. Equation (25.2.8) yields, for each feature point, two quadratic constraints on the two unknowns u and v. Although this system should *a priori* admit four solutions, it admits, as shown in the exercises, exactly two real solutions. In fact, given n point correspondences and the image positions of the three tie points, it can also be shown [**?**] that the pictures of the remaining $n-3$ points can be determined in closed form up to a two-fold ambiguity.

Once the positions of all feature points have been determined, the scene can be rendered by triangulating these points and texture mapping the triangles. Interestingly, hidden-surface removal can also be performed via traditional z-buffer techniques even though no explicit three-dimensional reconstruction is performed: the idea is to assign relative depth values to the vertices of the triangulation, and it is closely related to the method used in the affine structure from motion theorem from Chapter 13. Let Π denote the image plane of one of our input images, and Π' the image plane of our synthetic image. To render correctly two points P and Q that project onto the same point r' in the synthetic image, we must compare their depths (Figure 25.14).

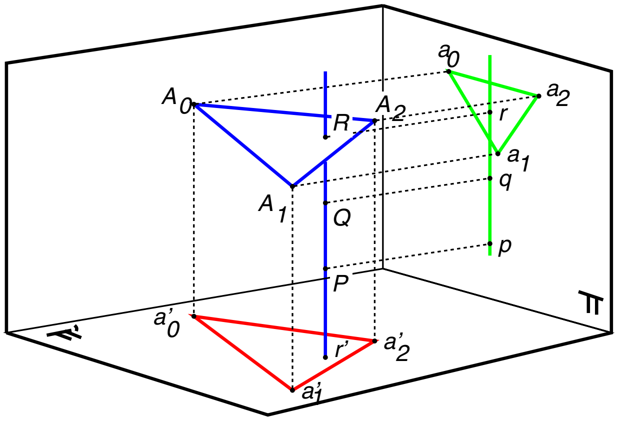


Figure 25.14. Z-buffering. Reprinted from [**?**], Figure X.

Let R denote the intersection of the viewing ray joining P to Q with the plane spanned by the reference points A_0, A_1 and A_2, and let p, q, r denote the projections of P, Q and R into the reference image. Suppose for the time being that P and Q are two of the points tracked in the input image; it follows that the positions of p and q are known. The position of r is easily computed by remarking that its coordinates in the affine basis of Π formed by the projections a_0, a_1, a_2 of the reference points are the same as the coordinates of R in the affine basis formed by the points A_0, A_1, A_2 in their own plane, and thus are also the same as the coordinates of r' in the affine basis of Π' formed by the projections a'_0, a'_1, a'_2 of the reference points.

The ratio of the depths of P and Q relative to the plane Π is simply the ratio $\overline{pr}/\overline{qr}$. Not that deciding which point is actually visible requires orienting the line supporting the points p, q, r, which is simply the epipolar line associated with the point r'. A coherent orientation should be chosen for all epipolar lines (this is easy since they are all parallel to each other). Note that this does not require explicitly computing the epipolar geometry: given a first point p', one can orient the line pr, then use the same orientation for all other point correspondences. The orientations chosen should also be consistent over successive frames, but this is not a problem since the direction of the epipolar lines changes slowly from one frame to the next, and one can simply choose the new orientation so that it makes an acute angle with the previous one.

The parameterized image variety approach to image-based rendering as presented above was implemented in [**?**] and examples of synthetic pictures constructed using this method are shown in Figure 25.15. Movies can be found in the CD accompanying this book.

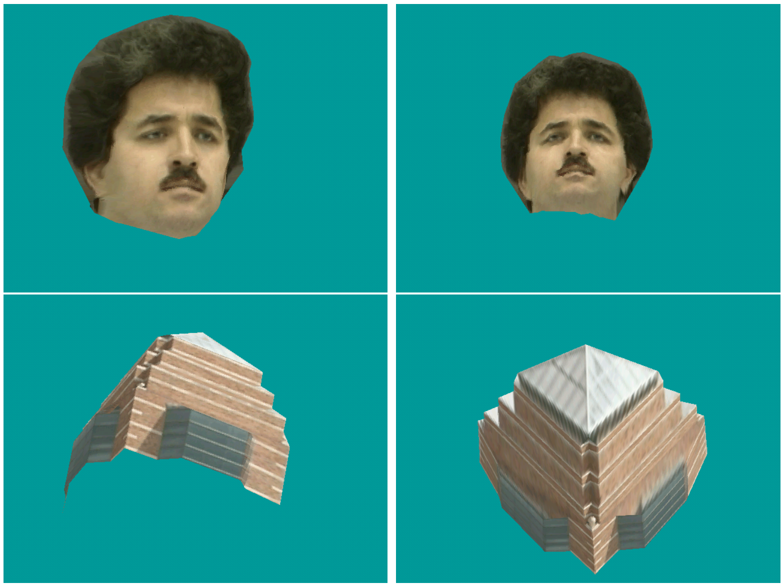


Figure 25.15. Images synthesized using parameterized image varieties.

25.3 The Light Field

This section discusses a very different approach to image-based rendering, whose only similarity with the techniques discussed in the previous section is that, like them, it does not require the construction of any implicit or explicit 3D model of a scene. Let us consider for example a panoramic camera that optically records the radiance along rays passing through a single point and covering a full hemisphere (see, for example, [**?**] and Figure 25.16(a)). It is possible to create any image observed by a virtual camera whose pinhole is located at this point by mapping the original image rays onto virtual ones. This allows a user to arbitrarily pan and tilt the virtual camera and interactively explore his or her visual environment. Similar effects can be obtained by stitching together close-by images taken by a hand-held camcorder into a mosaic (see, for example, [?] and Figure 25.16(b)), or by combining the pictures taken by a camera panning (and possibly tilting) about its optical center into a cylindrical mosaic (see, for example, [?] and Figure 25.16(c)).

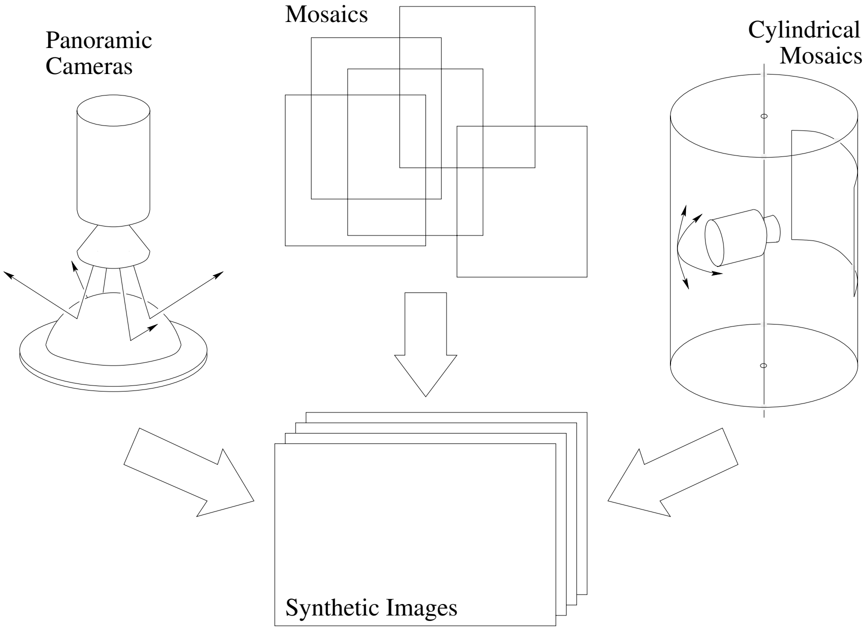


Figure 25.16. Constructing synthetic views of a scene from a fixed viewpoint.

These techniques have the drawback of limiting the viewer motions to pure rotations about the optical center of the camera. A more powerful approach can be devised by considering the *plenoptic function* [?] that associates with each point in space the (wavelength-dependent) radiant energy along a ray passing through this point at a given time (Figure 25.17(a)). The *light field* [?] is a snapshot of the plenoptic function for light travelling in vacuum in the absence of obstacles. This relaxes the dependence of the radiance on time and on the position of the point of interest along the corresponding ray (since radiance is constant along straight lines in a non-absorbing medium), and yields a representation of the plenoptic function by the radiance along the four-dimensional set of light rays. These rays can be parameterized in many different ways, e.g., using the Plücker coordinates introduced in Chapter 5, but a convenient parametrization in the context of image-based rendering is the *light slab*, where each ray is specified by the coordinates of its intersections with two arbitrary planes (Figure 25.17(b)).

The light slab is the basis for a two-stage approach to image-based rendering: during the learning stage, many views of a scene are used to create a discrete version

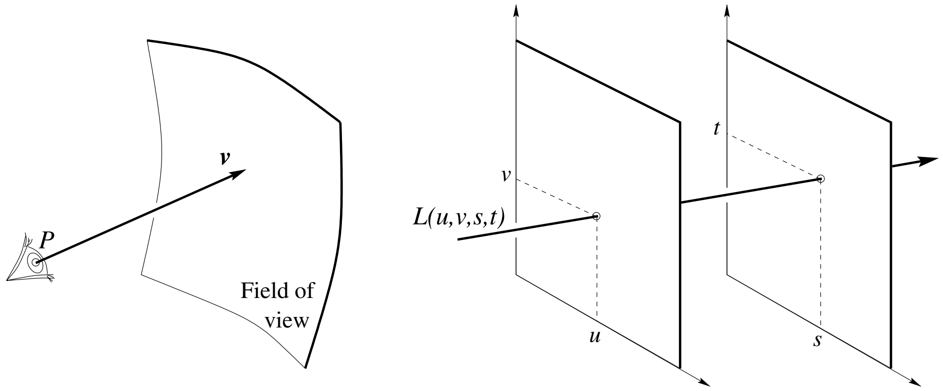


Figure 25.17. The plenoptic function and the light field: (left) the plenoptic function can be parameterized by the position P of the observer and the viewing direction $\boldsymbol{v}$; (right) the light field can be parameterized by the four parameters u, v, s, t defining a light slab. In practice, several light slabs are necessary to model a whole object and obtain full spherical coverage.

of the slab that can be thought of as a four-dimensional lookup table. At synthesis time, a virtual camera is defined and the corresponding view is interpolated from the lookup table. The quality of the synthesized images depends of course on the number of reference images. The closer the virtual view is to the reference images, the better the quality of the synthesized image. Note that constructing the light slab model of the light field does not require establishing correspondences between images. Figure 25.18 shows two aspects of a light slab constructed from a set of synthetic pictures, where the pinhole of the (virtual) camera is constrained to lie in the (u, v) plane: Figure 25.18(a) shows a 2D slice of a slab, corresponding to a given (u, v) location, or equivalently to a given image of the scene. Figure 25.18(b), on the other hand, shows the slice associated with a given (s, t) sample, or equivalently, a given scene point. This slab slice can also be thought of as the 2D slice of the BRDF corresponding to the illumination pattern used during image acquisition. In fact, it should be noted that, unlike most other methods for image-based rendering, that rely on texture mapping and thus assume (implicitly) that the observed surfaces are Lambertian, light-field techniques can be used to render (under a fixed illumination) pictures of objects with *arbitrary* BRDFs.

In practice, a sample of the light field is acquired by taking a large number of images and mapping pixel coordinates onto slab coordinates. Figure 25.19 illustrates the general case: the mapping between any pixel in the (x, y) image plane and the corresponding areas of the (u, v) and (s, t) plane defining a light slab is a planar projective transformation. Hardware- or sotware-based texture mapping can thus be used to populate the light field on a four-dimensional rectangular grid. In the experiments described in [**?**], light slabs are acquired in the simple setting of a camera mounted on a planar gantry and equipped with a pan-tilt head so it

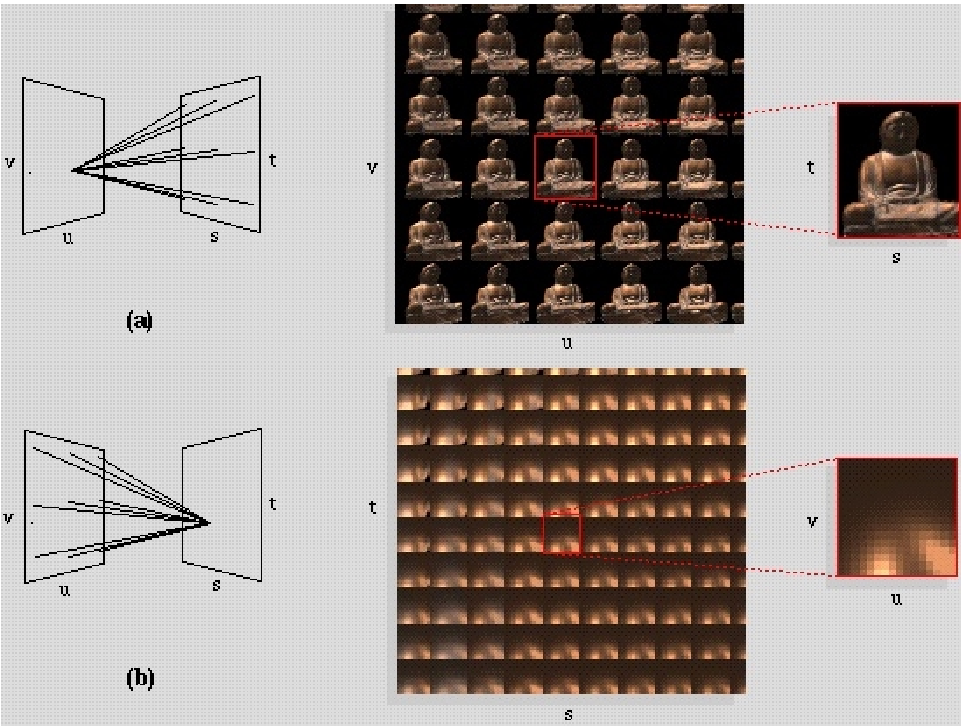


Figure 25.18. Two aspects of a light slab. See text for details. Reprinted from [?], Figure 6.

can rotate about its optical center and always point toward the center of the object of interest. In this context, all calculations can be simplified by taking, as in Figure 25.18, the (u, v) plane to be the plane in which the camera's optical center is constrained to remain.

At rendering time, the projective mapping between the (virtual) image plane and the two planes defining the light slab can once again be used to efficiently synthesize new images. Figure 25.20 shows sample pictures generated using the light field approach. The top three image pairs were generated using synthetic pictures of various objects to populate the light field. The last pair of images was constructed by using the planar gantry mentioned earlier to acquire 2048 256×256 images of a toy lion, grouped into four 32×16 light slabs.

An important issue is the size of the light slab representation: the raw input images of the lion take 402MB of disk space. There is of course much redundancy in these pictures, as in the case of successive frames in a motion sequence. A simple but effective two-level approach to image (de)compression is proposed in [**?**]: the four-dimensional light slab is first decomposed into 2^4 tiles of color values. These tiles are encoded using *vector quantization* [**?**], a lossy compression technique where

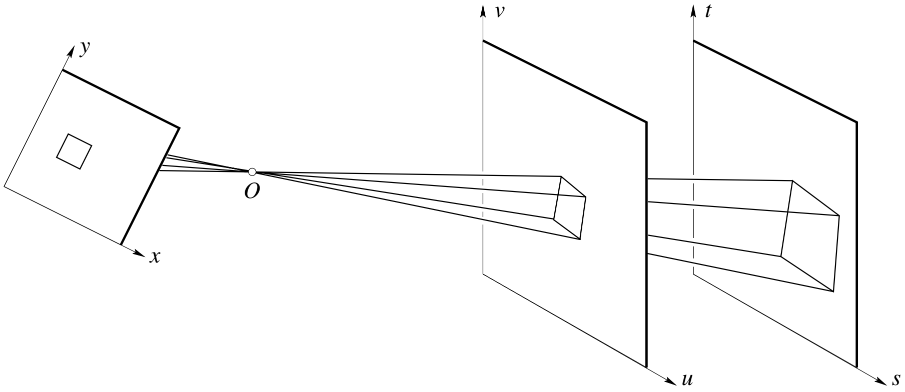


Figure 25.19. The acquisition of a light slab from images and the synthesis of new images from a light slab can be modeled via projective transformations between the (x, y) image plane and the (u, v) and (s, t) planes defining the slab.

the 48-dimensional vectors representing the original tiles are replaced by a relatively small set of reproduction vectors, called *codewords*, that best approximate in the mean-squared-error sense the input vectors. The light slab is thus represented by a set of indices in the *codebook* formed by all codewords. In the case of the lion, the codebook is relatively small (0.8MB) and the size of the set of indices is 16.8MB. The second compression stage consists of applying the *gzip* implementation of *entropy coding* [**?**] to the codebook and the indices. The final size of the representation is only 3.4MB, corresponding to a compression rate of 118:1. At rendering time, entropy decoding is performed as the file is loaded in main memory. Dequantization is performed on demand during display, and it allows interactive refresh rates.

25.4 Notes

Image-based rendering is a quickly expanding field. To close this chapter, let us just mention a few alternatives to the approaches already mentioned in the previous sections.

Variants of the volumetric approach to object modeling from registered silhouettes presented in Section 25.1 use polyhedra [**?**] or octrees [**?**] to represent the cones and their intersection, and include a commercial system, Sphinx3D [**?**], for automatically constructing polyhedral models from images. See also [**?**] for related work. The tangency constraint has also been used in various approaches for reconstructing a surface from a continuous sequence of outlines under known or unkown camera motions [**?**; **?**; **?**; **?**; **?**; **?**; **?**]. Variants of the view interpolation method discussed in Section 25.1 include [**?**; **?**; **?**].

Transfer-based approaches to image-based rendering include, besides those discussed in Section 25.2, [**?**; **?**].

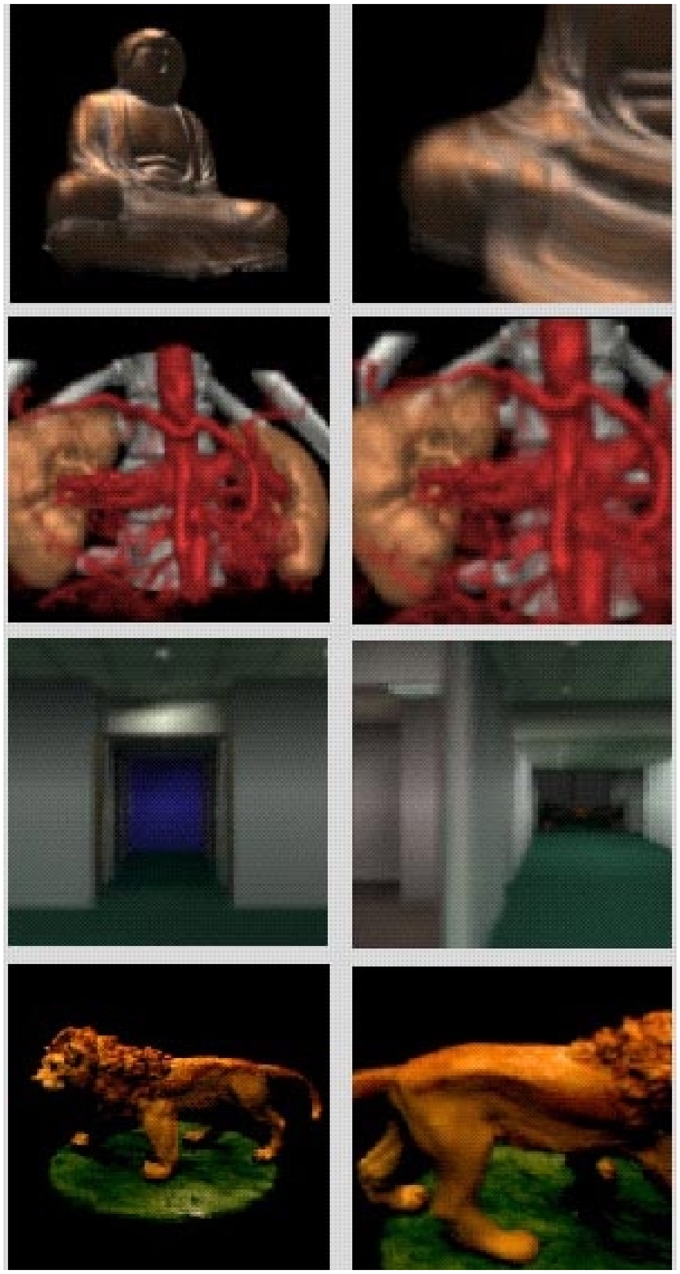


Figure 25.20. Images synthesized with the light field approach. Reprinted from [**?**], Figure 15.

As briefly mentioned in Section 25.3, a number of techniques have been developed for interactively exploring a user's visual environment from a fixed viewpoint. These include a commercial system, *QuickTime VR*, developed at Apple by Chen [**?**], and algorithms that reconstruct pinhole perspective images from panoramic pictures acquired by special-purpose cameras (see, for example, [**?**]). Similar effects can be obtained in a less controlled setting by stitching together close-by images taken by a hand-held camcorder into a mosaic (see, for example, [**?**]). Variants of the light field approach discussed in Section 25.3 include [**?**; **?**].

25.5 Assignments

Exercises

1. De Casteljeau construction of Bézier curves.
2. Degree elevation of a Bézier curve.
3. Structure and motion from silhouettes: the orthographic case in the plane.
4. Show that the construction of the points Q_i in Section 25.1.1 places these points in a plane that passes through the centroid O of the points C_i
5. Differentiation of error functions defined implicitly.
6. Computing an error measure between imagee edges and predicted lines: show that the integral of the signed distance h introduced in Section 25.1 over the edge where it is defined is equal to

$$E = \int_0^1 h^2(t)dt = \frac{l}{3}(h(0)^2 + h(0)h(1) + h(1)^2).$$

7. Linear methods for computing initial model parameters in the Façade system when some of the edge orientations are known.
8. Show that the set of all projective images of a fixed scenes is an eleven-dimensional variety.
9. Show that the set of all perspective images of a fixed scene (for a camera with constant intrinsic parameters) is a six-dimensional variety.
10. In this exercise we show that (25.2.8) only admits two solutions.
    (a) Show that (25.2.7) can be rewritten as

$$\begin{cases} X^2 - Y^2 + e_1 - e_2 = 0, \\ 2XY + e = 0, \end{cases} \tag{25.5.1}$$

where

$$\begin{cases} X = u + \alpha u_1 + \beta u_2, \\ Y = v + \alpha v_1 + \beta v_2, \end{cases}$$

and e, e_1 and e_2 are coefficients depending on u_1, v_1, u_2, v_2 and the structure parameters.

(b) Show that the variables X and Y defined by (25.5.1) are the u and v coordinates of the *affine motion field*.

(c) Show that the solutions of (25.5.1) are given by

$$\begin{cases} X' = \sqrt[4]{(e_1 - e_2)^2 + e^2} \cos(\frac{1}{2} \arctan(e, e_1 - e_2)), \\ Y' = \sqrt[4]{(e_1 - e_2)^2 + e^2} \sin(\frac{1}{2} \arctan(e, e_1 - e_2)), \end{cases}$$

and $(X'', Y'') = (-X', -Y')$. (Hint: use a change of variables to rewrite (25.5.1) as a system of trigonometric equations.)

BIBLIOGRAPHY

[Adelson and Bergen, 1991] E. Adelson and J. Bergen. The plenoptic function and the elements of early vision. In M. Landy and J. Movshon, editors, *Computational Models of Visual Processing.* MIT Press, Cambridge, MA, 1991.

[Agin, 1972] G.J. Agin. *Representation and description of curved objects.* PhD thesis, Stanford University, Stanford, CA, 1972.

[Aho *et al.*, 1974] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[Ahuja and Abbott, 1993] N. Ahuja and A.L. Abbott. Active stereo: Integrating disparity, vergence, focus, aperture, and calibration for surface estimation. *IEEE Trans. Patt. Anal. Mach. Intell.*, 15(10):1007–1029, 1993.

[Aloimonos *et al.*, 1987] Y. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. *Int. J. of Comp. Vision*, 1(4):333–356, 1987.

[Amenta *et al.*, 1998] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Siggraph '98*, pages 415–421, 1998.

[Anderson and Nayakama, 1994] B.L. Anderson and K. Nayakama. Toward a general theory of stereopsis - binocular matching, occluding contours, and fusion. *Psychological review*, 101(3):414–445, 1994.

[Arbogast and Mohr, 1991] E. Arbogast and R. Mohr. 3D structure inference from image sequences. *Journal of Pattern Recognition and Artificial Intelligence*, 5(5), 1991.

[Arend and Reeves, 1986] L.E. Arend and A. Reeves. Simultaneous colour constancy. *J. Opt. Soc. America - A*, 3:1743–1751, 1986.

[Armitage and Enser, 1997] L.H. Armitage and P.G.B. Enser. Analysis of user need in image archives. *Journal of Information Science*, 23(4):287–299, 1997.

[Arnol'd, 1984] V.I. Arnol'd. *Catastrophe Theory.* Springer-Verlag, Heidelberg, 1984.

[Arnon *et al.*, 1984] D.S. Arnon, G. Collins, and S. McCallum. Cylindrical algebraic decomposition I and II. *SIAM J. Comput.*, 13(4):865–889, November 1984.

[Avidan and Shashua, 1997] S. Avidan and A. Shashua. Novel view synthesis in tensor space. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 1034–1040, San Juan, Puerto Rico, 1997.

[Ayache and Faugeras, 1986] N. Ayache and O. Faugeras. Hyper: a new approach for the recognition and positioning of two-dimensional objects. *IEEE Trans. Patt. Anal. Mach. Intell.*, 8(1):44–54, January 1986.

[Ayache and Faverjon, 1997] N. Ayache and B. Faverjon. Efficient registration of stereo images by matching graph descriptions of edge segments. *Int. J. of Comp. Vision*, pages 101–137, 1997.

[Ayache and Lustman, 1987] N. Ayache and F. Lustman. Fast and reliable passive trinocular stereovision. In *Proc. Int. Conf. Comp. Vision*, pages 422–427, 1987.

[Bajcsy and Solina, 1987] R. Bajcsy and F. Solina. Three-dimensional object representation revisited. In *Proc. Int. Conf. Comp. Vision*, pages 231–240, London, U.K., June 1987.

[Bajcsy, 1988] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, 1988.

[Baker and Binford, 1981] H.H. Baker and T.O. Binford. Depth from edge- and intensity-based stereo. In *Proc. International Joint Conference on Artificial Intelligence*, pages 631–636, 1981.

[Barrow and Tenenbaum, 1981] H.G. Barrow and J.M. Tenenbaum. Interpreting line drawings of three-dimensional surfaces. *Artificial Intelligence Journal*, 17(1-3):75–116, 1981.

[Baumgart, 1974] B.G. Baumgart. Geometric modeling for computer vision. Technical Report AIM-249, Stanford University, 1974. Ph.D. Thesis. Department of Computer Science.

[Beardsley *et al.*, 1997] P. Beardsley, A. Zisserman, and D. Murray. Sequential updating of projective and affine structure from motion. *Int. J. of Comp. Vision*, 23(3):235–259, 1997.

[Belongie *et al.*, 1998] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color and texture-based image segmentation using em and its applications to content based image retrieval. In *Int. Conf. on Computer Vision*, 1998.

[Berger, 1987] M. Berger. *Geometry.* Springer-Verlag, 1987.

[Besl and Jain, 1988] P.J. Besl and R.C. Jain. Segmentation through variable-order surface fitting. *IEEE Trans. Patt. Anal. Mach. Intell.*, 10(2):167–192, March 1988.

[Besl and McKay, 1992] P.J. Besl and N.D. McKay. A method for registration of 3D shapes. *IEEE Trans. Patt. Anal. Mach. Intell.*, 14(2):239–256, 1992.

[Besl, 1989] P.J. Besl. Active optical range imaging sensors. *Machine vision and applications*, 1:127–152, 1989.

[Binford, 1984] T.O. Binford. Stereo vision: complexity and constraints. In *Int. Symp. on Robotics Research*, pages 475–487. MIT Press, 1984.

[Bishop, 1995] C. Bishop. *Neural networks for pattern recognition.* Oxford University Press, 1995.

[Blake and Brelstaff, 1987] A. Blake and G. Brelstaff. Computing lightness. *Pattern Recognition Letters*, 5:129–138, 1987.

[Blake, 1985] A. Blake. Boundary conditions for lightness computation in mondrian world. *Computer Vision, Graphics and Image Processing*, 32:314–327, 1985.

[Boissonnat and Germain, 1981] J.-D. Boissonnat and F. Germain. A new approach to the problem of acquiring randomly-oriented workpieces from a bin. In *Proc. International Joint Conference on Artificial Intelligence*, Vancouver, Canada, 1981.

[Boissonnat, 1984] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transaction on Computer Graphics*, 3(4):266–286, 1984.

[Boreczky and Rowe, 1996] J.S. Boreczky and L.A. Rowe. Comparison of video shot boundary detection techniques. *Journal of Electronic Imaging*, 5(2):122–8, 1996.

[Boult and Brown, 1991] T.E. Boult and L.G. Brown. Factorization-based segmentation of motions. In *IEEE Workshop on Visual Motion*, pages 179–186, 1991.

[Boyer and Berger, 1996] E. Boyer and M.-O. Berger. 3d surface reconstruction using occluding contours. *Int. J. of Comp. Vision*, 1996. To appear.

[Boyer, 1996] E. Boyer. Object Models from Contour Sequences. In *Proceedings of Fourth European Conference on Computer Vision, Cambridge, (England)*, pages 109–118, April 1996. Lecture Notes in Computer Science, volume 1065.

[Brady *et al.*, 1985] J.M. Brady, J. Ponce, A. Yuille, and H. Asada. Describing surfaces. *Computer Vision, Graphics and Image Processing*, 32(1):1–28, August 1985.

[Brelstaff, 1988] G. Brelstaff. Detecting specular reflections using lambertian constraints. In *Int. Conf. on Computer Vision*, 1988.

[Bruce *et al.*, 1996a] J.W. Bruce, P.J. Giblin, and F. Tari. Parabolic curves of evolving surfaces. *Int. J. of Comp. Vision*, 17(3):291–306, 1996.

[Bruce *et al.*, 1996b] J.W. Bruce, P.J. Giblin, and F. Tari. Ridges, crests and subparabolic lines of evolving surfaces. *Int. J. of Comp. Vision*, 18(3):195–210, 1996.

[Brunie *et al.*, 1992] L. Brunie, S. Lavallée, and R. Szeliski. Using force fields derived from 3D distance maps for inferring the attitude of a 3D rigid object. In G. Sandini, editor, *Proc. European Conf. Comp. Vision*, volume 588 of *Lecture Notes in Computer Science*, pages 670–675. Springer-Verlag, 1992.

[Brunnström *et al.*, 1996] K. Brunnström, J.-O. Ekhlund, and T. Uhlin. Active fixation for scene exploration. *Int. J. of Comp. Vision*, 17(2):137–162, 1996.

[Buchsbaum, 1980] G. Buchsbaum. A spatial processor model for object colour perception. *J. Franklin Inst.*, 310:1–26, 1980.

[Burl *et al.*, 1995] M.C. Burl, T.K. Leung, and P. Perona. Face localisation via shape statistics. In *Int. Workshop on Automatic Face and Gesture Recognition*, 1995.

[Callahan and Weiss, 1985] J. Callahan and R. Weiss. A model for describing surface shape. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 240–245, San Francisco, CA, June 1985.

[Canny, 1988] J.F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, 1988.

[Carmichael *et al.*, 1999] O. Carmichael, D.F. Huber, and M. Hebert. Large data sets and confusing scenes in 3-D surface matching and recognition. In *Second International Conference on 3-D Digital Imaging and Modeling (3DIM'99)*, pages 358–367, 1999.

[Carson and Ogle, 1996] C. Carson and V. E. Ogle. Storage and retrieval of feature data for a very large online image collection. In *IEEE Computer Society Bulletin of the Technical Committee on Data Engineering*, 1996.

[Carson *et al.*, 1997] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. In *Workshop on Content Based Access of Image and Video Libraries*, 1997. Workshop at IEEE conference on Computer Vision and Pattern Recognition.

[Carson *et al.*, 1999] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third Int. Conf. on Visual Information Systems*, 1999. Will appear.

[Cascia *et al.*, 1998] M. La Cascia, S. Sethi, and S. Sclaroff. Combining textual and visual cues for content based image retrieval on the web. In *IEEE Workshop on Content Based Access of Image and Video Libraries*, pages 24–28, 1998.

[Castore, 1984] G. Castore. Solid modeling, aspect graphs, and robot vision. In Pickett and Boyse, editors, *Solid modeling by computer*, pages 277–292. Plenum Press, NY, 1984.

[Chakravarty, 1982] I. Chakravarty. The use of characteristic views as a basis for recognition of three-dimensional objects. Image Processing Laboratory IPL-TR-034, Rensselaer Polytechnic Institute, October 1982.

[Chang *et al.*, 1997a] S.-F. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. Videoq- an automatic content-based video search system using visual cues. In *ACM Multimedia Conference*, 1997.

[Chang *et al.*, 1997b] S-F. Chang, J.R. Smith, M. Beigi, and A. Benitez. Visual information retrieval from large distributed online repositories. *Comm. ACM*, 40(12):63–71, 1997.

[Chang *et al.*, 1998a] S.-F. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. A fully automated content based video search engine supporting spatiotemporal queries. *IEEE Trans. Circuits and Systems for Video Technology*, 8(8):602–615, 1998.

[Chang *et al.*, 1998b] S-F. Chang, W. Chen, and H. Sundaram. Semantic visual templates - linking visual features to semantics. In *IEEE Int. Conf. Image Processing*, 1998.

[Chapelle *et al.*, 1999] O. Chapelle, P. Haffner, and V. Vapnik. Svm's for histogram-based image classification. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 1999. In review.

[Chasles, 1855] M. Chasles. Question no. 296. *Nouv. Ann. Math.*, 14(50), 1855.

[Chen and Freeman, 1991] S. Chen and H. Freeman. On the characteristic views of quadric-surfaced solids. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 34–43, June 1991.

[Chen, 1995] S.E. Chen. Quicktime VR: An image-based approach to virtual environment navigation. In *SIGGRAPH*, pages 29–38, Los Angeles, CA, August 1995.

[Chiyokura and Kimura, 1983] B. Chiyokura and F. Kimura. Design of solids with free-form surfaces. *Computer Graphics*, 17(3):289–298, 1983.

[Christy and Horaud, 1996] S. Christy and R. Horaud. Euclidean shape and motion from multiple perspective views by affine iterations. *IEEE Trans. Patt. Anal. Mach. Intell.*, 18(11):1098–1104, 1996.

[Chua and Jarvis, 1996] C. Chua and R. Jarvis. Point signatures: a new representation for 3D object recognition. *Int. J. of Comp. Vision*, 25(1):63–85, 1996.

[Cipolla and Blake, 1992] R. Cipolla and A. Blake. Surface shape from the deformation of the apparent contour. *Int. J. of Comp. Vision*, 9(2):83–112, November 1992.

[Cipolla *et al.*, 1995] R. Cipolla, K.E. Astrom, and P.J. Giblin. Motion from the frontier of curved surfaces. In *Proc. Int. Conf. Comp. Vision*, pages 269–275, Boston, MA, 1995.

[Clarkson, 1988] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Computing*, 17:830–847, 1988.

[Clowes, 1971] M.B. Clowes. On seeing things. *Artificial Intelligence Journal*, 2(1):79–116, 1971.

[Cohen, 1964] J. Cohen. Dependency of the spectral reflectance curves of the munsell color chips. *Psychon. Sci.*, 1:369–370, 1964.

[Collins, 1971] G.E. Collins. The calculation of multivariate polynomial resultants. *Journal of the ACM*, 18(4):515–522, Oct 1971.

[Collins, 1975] G.E. Collins. *Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition*, volume 33 of *Lecture Notes in Computer Science.* Springer-Verlag, New York, 1975.

[Congiu *et al.*, 1995] G. Congiu, A. Del Bimbo, and E. Vicario. Iconic retrieval by contents from databases of cardiological sequences. In *Visual Database Systems 3: Proc third IFIP 2.6 working conference on visual database systems*, pages 158–74, 1995.

[Connolly and Stenstrom, 1989] C.I. Connolly and J.R. Stenstrom. 3D scene reconstruction from multiple intensity images. In *Proc. IEEE Workshop on Interpretation of 3D Scenes*, pages 124–130, Austin, TX, November 1989.

[Costeira and Kanade, 1998] J. Costeira and T. Kanade. A multi-body factorization method for motion analysis. *Int. J. of Comp. Vision*, 29(3):159–180, September 1998.

[Coxeter, 1974] H.S.M. Coxeter. *Projective Geometry.* Springer-Verlag, 1974. Second Edition.

[Craig, 1989] J.J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley, 1989. Second edition.

[Cross *et al.*, 1999] G. Cross, A.W. Fitzgibbon, and A. Zisserman. Parallax geometry of smooth surfaces in multiple views. In *Proc. Int. Conf. Comp. Vision*, pages 323–329, Corfu, Greece, 1999.

[Curless and Levoy, 1996] B. Curless and M. Levoy. A volumetric method for building complex images from range images. In *SIGGRAPH*, New Orleans, LA, August 1996.

[Cutler and Davis, 2000] Ross Cutler and Larry S. Davis. Robust real-time periodic motion detection, analysis and applications. *IEEE T. Pattern Analysis and Machine Intelligence*, 22:781–796, 2000.

[De Bonet and Viola, 1997] J. S. De Bonet and P. Viola. Structure driven image database retrieval. In *Advances in Neural Information Processing*, volume 10, 1997.

[Debevec *et al.*, 1996] P. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH*, pages 11–20, New Orleans, LA, August 1996.

[Demazure, 1989] M. Demazure. *Catastrophes et Bifurcations.* Editions Ellipses, 1989.

[Deutscher *et al.*, 2000] Jonathan Deutscher, Andrew Blake, and Ian Reid. Articulated body motion capture by annealed particle filtering. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2000.

[Devernay and Faugeras, 1994] F. Devernay and O.D. Faugeras. Computing differential properties of 3D shapes from stereopsis without 3D models. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 208–213, Seattle, WA, June 1994.

[Devroye *et al.*, 1996] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer Verlag, 1996.

[do Carmo, 1976] M.P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Dove, 1841] H.W. Dove. Über Stereoskopie. *Annals Phys. Series 2*, 110:494–498, 1841.

[Draper, 1981] S.W. Draper. The use of gradient and dual space in line-drawing interpretation. *Artificial Intelligence Journal*, 17:461–508, 1981.

[Duda and Hart, 1973] R. Duda and P. Hart. *Pattern classification and scene analysis.* Wiley, 1973.

[D'Zmura and Lennie, 1986] M. D'Zmura and P. Lennie. Mechanisms of colour constancy. *J. Opt. Soc. America - A*, 3:1662–1672, 1986.

[Eakins *et al.*, 1998] J.P. Eakins, J.M. Boardman, and M.E. Graham. Similarity retrieval of trademark images. *IEEE Multimedia*, 5(2):53–63, 1998.

[Eggert and Bowyer, 1989] D. Eggert and K. Bowyer. Computing the orthographic projection aspect graph of solids of revolution. In *Proc. IEEE Workshop on Interpretation of 3D Scenes*, pages 102–108, Austin, TX, November 1989.

[Eggert and Bowyer, 1991] D. Eggert and K. Bowyer. Perspective projection aspect graphs of solids of revolution: An implementation. In *IEEE Workshop on Directions in Automated CAD-Based Vision*, pages 44–53, June 1991.

[Eggert *et al.*, 1993] D. Eggert, K. Bowyer, C. Dyer, H. Christensen, and D. Goldgof. The scale space aspect graph. *IEEE Trans. Patt. Anal. Mach. Intell.*, 15(11):1114–1130, 1993.

[Eggert, 1991] D. Eggert. *Aspect Graphs of Solids of Revolution.* PhD thesis, University of South Florida, December 1991.

[Enser, 1993] P.G.B. Enser. Query analysis in a visual information retrieval context. *J. Document and Text Management*, 1(1):25–52, 1993.

[Enser, 1995] P.G.B. Enser. Pictorial information retrieval. *Journal of documentation*, 51(2):126–170, 1995.

[Fan *et al.*, 1987] T.J. Fan, G. Médioni, and R. Nevatia. Segmented descriptions of 3D surfaces. *IEEE Transactions on Robotics and Automation*, 3(6):527–538, December 1987.

[Farin, 1993] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego, CA, 1993.

[Faugeras and Hebert, 1986] O.D. Faugeras and M. Hebert. The representation, recognition, and locating of 3-D objects. *International Journal of Robotics Research*, 5(3):27–52, Fall 1986.

[Faugeras and Mourrain, 1995] O. Faugeras and B. Mourrain. On the geometry and algebra of the point and line correspondences between n images. Technical Report 2665, INRIA Sophia-Antipolis, 1995.

[Faugeras and Papadopoulo, 1997] O. Faugeras and T. Papadopoulo. Gaussman-Caylay algebra for modeling systems of cameras and the algebraic equations of the manifold of trifocal tensors. Technical Report 3225, INRIA Sophia-Antipolis, 1997.

[Faugeras *et al.*, 1992] O.D. Faugeras, Q.-T. Luong, and S.J. Maybank. Camera self-calibration: theory and experiments. In G. Sandini, editor, *Proc. European Conf. Comp. Vision*, volume 588 of *Lecture Notes in Computer Science*, pages 321–334, Santa Margherita, Italy, 1992. Springer-Verlag.

[Faugeras, 1992] O.D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In G. Sandini, editor, *Proc. European Conf. Comp. Vision*, volume 588 of *Lecture Notes in Computer Science*, pages 563–578, Santa Margherita, Italy, 1992. Springer-Verlag.

[Faugeras, 1993] O.D. Faugeras. *Three-Dimensional Computer Vision.* MIT Press, 1993.

[Faugeras, 1995] O.D. Faugeras. Stratification of 3D vision: projective, affine and metric representations. *J. Opt. Soc. Am. A*, 12(3):465–484, March 1995.

[Fitzgibbon and Zisserman, 1998] A. Fitzgibbon and A. Zisserman. Automatic 3D model acquisition and generation of new images from video sequences. In *European Signal Processing Conference*, pages 311–326, Rhodes, Greece, 1998.

[Flickner *et al.*, 1995] M. Flickner, H. Sawhney, W. Niblack, and J. Ashley. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, 1995.

[Flock, 1984] H.R. Flock. Illumination: inferred or observed? *Perception and psychophysics*, 35, 1984.

[Forney, 1973] G.D.Jr. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3), March 1973.

[Forsyth and Fleck, 1997] D.A. Forsyth and M.M. Fleck. Body plans. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 1997.

[Forsyth and Fleck, 1999] D.A. Forsyth and M.M. Fleck. Automatic detection of human nudes. *Int. J. Computer Vision*, 32, 1999.

[Forsyth *et al.*, 1996] D.A. Forsyth, M.M. Fleck, and C. Bregler. Finding naked people. In *European Conference on Computer Vision*, 1996.

[Forsyth, 99] D. A. Forsyth. Sampling, resampling and colour constancy. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 99. in review.

[Friedman *et al.*, 1977] J. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Math. Software*, 3(3), 1977.

[Frisby, 1980] John Frisby. *Seeing: Illusion, Brain and Mind.* Oxford University Press, 1980.

[Fukunaga, 1990] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[Gaston and Lozano-Pérez, 1984] P.C. Gaston and T. Lozano-Pérez. Tactile recognition and localization using object models: The case of polyhedra in the plane. *IEEE Trans. Patt. Anal. Mach. Intell.*, 6(3), 1984.

[Gear, 1994] C.W. Gear. Feature grouping in moving objects. In *IEEE Workshop on Motion of Nonrigid and Articulate Objects*, Austin, TX, November 1994.

[Genc and Ponce, 1998] Y. Genc and J. Ponce. Parameterized image varieties: A novel approach to the analysis and synthesis of image sequences. In *Proc. Int. Conf. Comp. Vision*, pages 11–16, Bombay, India, January 1998.

[Gennery, 1980] D.B. Gennery. *Modelling the environment of an exploring vehicle by means of stereo vision.* PhD thesis, Stanford University, Stanford, CA, 1980.

[Gersho and Gray, 1992] A. Gersho and R.M. Gray. *Vector quantization and signal compression.* Kluwer Academic Publishers, 1992.

[Gershon *et al.*, 1986] R. Gershon, A.D. Jepson, and J.K. Tsotsos. Ambient illumination and the determination of material changes. *J. Opt. Soc. America*, A-3(10):1700–1707, 1986.

[Gigus and Malik, 1990] Z. Gigus and J. Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Trans. Patt. Anal. Mach. Intell.*, 12(2):113–122, February 1990.

[Gigus *et al.*, 1991] Z. Gigus, J. Canny, and R. Seidel. Efficiently computing and representing aspect graphs of polyhedral objects. *IEEE Trans. Patt. Anal. Mach. Intell.*, 13(6), June 1991.

[Gortler *et al.*, 1996] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *SIGGRAPH*, pages 43–54, New Orleans, LA, August 1996.

[Grimson and Lozano-Pérez, 1987] W.E.L. Grimson and T. Lozano-Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. Patt. Anal. Mach. Intell.*, 9(4):469–482, 1987.

[Grimson, 1981a] W.E.L. Grimson. A computer implementation of a theory of human stereo vision. *Philosophical Transactions of the Royal Society of London*, pages 217–253, 1981.

[Grimson, 1981b] W.E.L. Grimson. *From images to surfaces.* MIT Press, 1981.

[Gross and Boult, 1988] A.D. Gross and T.E. Boult. Error of fit measures for recovering parametric solids. In *Proc. Int. Conf. Comp. Vision*, pages 690–694, Tampa, FL, December 1988.

[Hamilton, 1844] W.R. Hamilton. On a new species of imaginary quantities connected with a theory of quaternions. *Transactions of the Royal Irish Academy*, 2:424–434, 1844.

[Hampapur *et al.*, 1997] A. Hampapur, A. Gupta, B. Horowitz, and Chiao-Fe Shu. Virage video engine. In *Storage and Retrieval for Image and Video Databases V – Proceedings of the SPIE*, volume 3022, pages 188–98, 1997.

[Hancock *et al.*, 1998] J. Hancock, D. Langer, M. Hebert, R. Sullivan, D. Ingimarson, E. Hoffman, M. Mettenleitner, and C. Froehlich. Active laser radar with high performance measurements. In *IEEE Int. Conf. on Robotics and Automation*, 1998.

[Hanrahan *et al.*, 1991] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. In *SIGGRAPH 91*, pages 197–206, 1991.

[Haralick and Shapiro, 1992] R.M. Haralick and L.G. Shapiro. *Computer and robot vision.* Addison Wesley, 1992.

[Haritaoglu *et al.*, 2000] Ismail Haritaoglu, David Harwood, and Larry S. Davis. W4: Real-time surveillance of people and their activities. *IEEE T. Pattern Analysis and Machine Intelligence*, 22:809–830, 2000.

[Hartley *et al.*, 1992] R.I. Hartley, R. Gupta, and T. Chang. Stereo from uncalibrated cameras. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 761–764, Champaign, IL, 1992.

[Hartley, 1994a] R.I. Hartley. An algorithm for self calibration from several views. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 908–912, Seattle, WA, June 1994.

[Hartley, 1994b] R.I. Hartley. Projective reconstruction and invariants from multiple images. *IEEE Trans. Patt. Anal. Mach. Intell.*, 16(10):1036–1041, 1994.

[Hartley, 1995] R.I. Hartley. In defence of the 8-point algorithm. In *Proc. Int. Conf. Comp. Vision*, pages 1064–1070, Boston, MA, 1995.

[Hartley, 1997] R. Hartley. Lines and points in three views and the trifocal tensor. *Int. J. of Comp. Vision*, 22(2):125–140, March 1997.

[Hartley, 1998] R. Hartley. Computation of the quadrifocal tensor. In *Proc. European Conf. Comp. Vision*, pages 20–35, 1998.

[Havaldar *et al.*, 1996] P. Havaldar, M.S. Lee, and G. Medioni. View synthesis from unregistered 2D images. In *Graphics Interface'96*, pages 61–69, 1996.

[Healey and Binford, 1986] G. Healey and T.O. Binford. Local shape from specularity. Computer Science STAN-CS-86-1139, Stanford University, June 1986.

[Hebert and Kanade, 1985] M. Hebert and T. Kanade. The 3D profile method for object recognition. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 458–463, San Francisco, CA, June 1985.

[Hebert, 2000] M. Hebert. Active and passive range sensing for robotics. In *IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, 2000.

[Helmholtz, 1909] H. von Helmholtz. *Physiological optics.* Dover, 1909. 1962 edition of the English translation of the 1909 German original, first published by the Optical Society of America in 1924.

[Hesse, 1863] O. Hesse. Die cubische Gleichung, von welcher die Lösung des Problems der Homographie von M. Chasles abhängt. *J. Reine Angew. Math.*, 62:188–192, 1863.

[Heyden and Åström, 1998] A. Heyden and K. Åström. Minimal conditions on intrinsic parameters for Euclidean reconstruction. In *Asian Conference on Computer Vision*, Hong Kong, 1998.

[Heyden and Åström, 1999] A. Heyden and K. Åström. Flexible calibration: minimal cases for auto-calibration. In *Proc. Int. Conf. Comp. Vision*, pages 350–355, Kerkyra, Greece, September 1999.

[Heyden, 1995] A. Heyden. *Geometry and algebra of multiple projective transformations.* PhD thesis, Lund University, Sweden, 1995.

[Heyden, 1998] A. Heyden. A common framework for multiple view tensors. In *Proc. European Conf. Comp. Vision*, pages 3–19, 1998.

[Hilbert and Cohn-Vossen, 1952] D. Hilbert and S. Cohn-Vossen. *Geometry and the Imagination.* Chelsea, New York, 1952.

[Holt and Hartwick, 1994a] B. Holt and L. Hartwick. 'quick, who painted fish?': searching a picture database with the qbic project at uc davis. *Information Services and Use*, 14(2):79–90, 1994.

[Holt and Hartwick, 1994b] B. Holt and L. Hartwick. Retrieving art images by image content: the uc davis qbic project. In *ASLIB Proceedings*, volume 46, pages 243–8, 1994.

[Horn, 1974] B. K. P. Horn. Determining lightness from an image. *Computer Vision, Graphics and Image Processing*, 3:277–299, 1974.

[Horn, 1986] B.K.P. Horn. *Computer Vision.* MIT Press, Cambridge, Mass., 1986.

[Horn, 1987] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, 4(4):629–642, April 1987.

[Huang and Faugeras, 1989] T.S. Huang and O.D. Faugeras. Some properties of the E-matrix in two-view motion estimation. *IEEE Trans. Patt. Anal. Mach. Intell.*, 11(12):1310–1312, December 1989.

[Huang and Zabih, 1998] J. Huang and R. Zabih. Combining color and spatial information for content-based image retrieval. In *European Conference on Digital Libraries*, 1998. web version at http://www.cs.cornell.edu/html/rdz/Papers/ECDL2/spatial.htm.

[Huang *et al.*, 1997a] C-Y. Huang, O.T. Camps, and T. Kanungo. Object recognition using appearance-based parts and relations. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 877–83, 1997.

[Huang *et al.*, 1997b] J. Huang, S.R. Kumar, M. Mitra, W-J. Zhu, and R. Zabih. Image indexing using color correlograms. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 762–768, 1997.

[Huffman, 1971] D.A. Huffman. Impossible objects as nonsense sentences. *Machine Intelligence*, 6:295–323, 1971.

[Huttenlocher and Ullman, 1987] D.P. Huttenlocher and S. Ullman. Object recognition using alignment. In *Proc. Int. Conf. Comp. Vision*, pages 102–111, London, U.K., June 1987.

[Ikeuchi and Kanade, 1988] K. Ikeuchi and T. Kanade. Automatic generation of object recognition programs. *Proceedings of the IEEE*, 76(8):1016–35, August 1988.

[Ingle, 1985] D.J. Ingle. The goldfish is a retinex animal. *Science*, 227:651–654, 1985.

[Ioffe and Forsyth, 1998] S. Ioffe and D.A. Forsyth. Learning to find pictures of people. In *NIPS*, 1998.

[Jacobsen and Gilchrist, 1988] A. Jacobsen and A. Gilchrist. The ratio principle holds over a million-to-one range of illumination. *Perception and Psychophysics*, 43:1–6, 1988.

[Jain and Vailaya, 1998] A.K. Jain and A. Vailaya. Shape-based retrieval: a case study with trademark image databases. *Pattern Recognition*, 31(9):1369–1390, 1998.

[Jarvis, 1983] R.A. Jarvis. A perspective on range finding techniques in computer vision. *IEEE Trans. Patt. Anal. Mach. Intell.*, 5(2):122–139, March 1983.

[Johnson and Hebert, 1998] A.E. Johnson and M. Hebert. Surface matching for object recognition in complex three-dimensional scenes. *Image and Vision Computing*, 16:635–651, 1998.

[Johnson and Hebert, 1999] A.E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. Patt. Anal. Mach. Intell.*, 21(5):433–449, 1999.

[Joshi *et al.*, 1999] T. Joshi, N. Ahuja, and J. Ponce. Structure and motion estimation from dynamic silhouettes under perspective projection. *Int. J. of Comp. Vision*, 31(1):31–50, February 1999.

[Judd, 1940] D.B. Judd. Hue, saturation and lightness of surface colors with chromatic illumination. *J. Opt. Soc. America*, 30:2–32, 1940.

[Julesz, 1960] B. Julesz. Binocular depth perception of computer-generated patterns. *The BellSystem Technical Journal*, 39(5):1125–1162, 1960.

[Julesz, 1971] B. Julesz. *Foundations of Cyclopean Perception.* The University of Chicago Press, London, 1971.

[Julesz, 1982] B. Julesz. Toward the automation of depth perception. In *IFIPS Congress*, Munich, Germany, 1982.

[Kajiya, 1982] J.T. Kajiya. Ray tracing parametric patches. *Computer Graphics*, 16:245–254, July 1982.

[Kanade and Narayanan, 1995] T. Kanade and P.W. Rander J.P Narayanan. Virtualized reality: Concepts and early results. In *IEEE Workshop on the Representation of Visual Scenes*, pages 69–76, 1995.

[Kanade *et al.*, 1997] T. Kanade, P.W. Rander, and J.P Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–47, 1997.

[Kanade *et al.*, 1998] T. Kanade, H. Saito, and S. Vedula. The 3D room: Digitizing time-varying 3D events by synchronized multiple video streams. Technical Report CMU-RI-TR98-34, Carnegie-Mellon University, 1998.

[Kanade, 1981] T. Kanade. Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence Journal*, 17:409–460, 1981.

[Kass, 1987] M. Kass. Linear image features in stereopsis. *Int. J. of Comp. Vision*, 1(4):357–368, 1987.

[Kato and Fujimura, 1989] T. Kato and K. Fujimura. Trademark: Multimedia image database system with intelligent human interface. *Denshi Joho Tsushin Gakkai Ronbunshi*, 72-DII(4):535–544, 1989. Translated in Systems and Computers in Japan, V21 n11, 33-45, 1990.

[Kato *et al.*, 1988] T. Kato, H. Shimogaki, T. Mizutori, and K. Fujimura. Trademark: Multimedia database with abstracted representation on knowledge base. In *Proc. Second Int Symp on Interoperable Information Systems*, pages 245–252, 1988.

[Kelly *et al.*, 1977] R.E. Kelly, P.R.H. McConnell, and S.J. Mildenberger. The Gestalt photomapping system. *Photogrammetric Engineering and Remote Sensing*, 43(11):1407–1417, 1977.

[Keren *et al.*, 1994] D. Keren, D. Cooper, and J. Subrahmonia. Describing complicated objects by implicit polynomials. *IEEE Trans. Patt. Anal. Mach. Intell.*, 16(1):38–53, 1994.

[Kergosien, 1981] Y.L. Kergosien. La famille des projections orthogonales d'une surface et ses singularités. *C.R. Acad. Sc. Paris*, 292:929–932, 1981.

[Klinker *et al.*, 1987] G.J. Klinker, S.A. Shafer, and T. Kanade. Using a colour reflection model to separate highlights from object colour. In *Int. Conf. on Computer Vision*, 1987.

[Koenderink and Van Doorn, 1976a] J.J. Koenderink and A.J. Van Doorn. Geometry of binocular vision and a model for stereopsis. *Biological Cybernetics*, 21:29–35, 1976.

[Koenderink and Van Doorn, 1976b] J.J. Koenderink and A.J. Van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, 24:51–59, 1976.

[Koenderink and Van Doorn, 1979] J.J. Koenderink and A.J. Van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[Koenderink and Van Doorn, 1990] J.J. Koenderink and A.J. Van Doorn. Affine structure from motion. *J. Opt. Soc. Am. A*, 8:377–385, 1990.

[Koenderink and Van Doorn, 1997] J.J. Koenderink and A.J. Van Doorn. The generic bilinear calibration-estimation problem. *Int. J. of Comp. Vision*, 23(3):217–234, July 1997.

[Koenderink, 1984] J.J. Koenderink. What does the occluding contour tell us about solid shape? *Perception*, 13:321–330, 1984.

[Koenderink, 1986] J.J. Koenderink. An internal representation for solid shape based on the topological properties of the apparent contour. In W. Richards and S. Ullman, editors, *Image Understanding: 1985-86*, chapter 9, pages 257–285. Ablex Publishing Corp., Norwood, NJ, 1986.

[Koenderink, 1990] J.J. Koenderink. *Solid Shape*. MIT Press, Cambridge, MA, 1990.

[Kriegman and Ponce, 1990] D.J. Kriegman and J. Ponce. Computing exact aspect graphs of curved objects: solids of revolution. *Int. J. of Comp. Vision*, 5(2):119–135, 1990.

[Krinov, 1947] E.L. Krinov. Spectral reflectance properties of natural formations. Technical report, National Research Council of Canada, Technical Translation: TT-439, 1947.

[Kruppa, 1913] E. Kruppa. Zur ermittung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss., Wien, Math. Naturw. Kl., Abt. Ila.*, 122:1939–1948, 1913.

[Kutulakos and Seitz, 1999] K.M. Kutulakos and S.M. Seitz. A theory of shape by space carving. In *Proc. Int. Conf. Comp. Vision*, pages 307–314, Corfu, Greece, 1999.

[Kutulakos and Vallino, 1998] K.N. Kutulakos and J. Vallino. Calibration-free augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):1–20, Jan–Mar 1998.

[Land and McCann, 1971] E.H. Land and J.J. McCann. Lightness and retinex theory. *J. Opt. Soc. Am.*, 61(1):1–11, 1971.

[Laveau and Faugeras, 1994] S. Laveau and O.D. Faugeras. 3D scene representation as a collection of images and fundamental matrices. Technical Report 2205, INRIA Sophia-Antipolis, 1994.

[Lee, 1986] H.C. Lee. Method for computing the scene-illuminant chromaticity from specular highlights. *J. Opt. Soc. Am.-A*, 3:1694–1699, 1986.

[Leung *et al.*, 1995] T.K. Leung, M.C. Burl, and P. Perona. Finding faces in cluttered scenes using random labelled graph matching. In *Int. Conf. on Computer Vision*, 1995.

[Levoy and Hanrahan, 1996] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH*, pages 31–42, New Orleans, LA, August 1996.

[Lim and Binford, 1988] H.S. Lim and T.O. Binford. Curved surface reconstruction using stereo correspondence. In *Proc. DARPA Image Understanding Workshop*, pages 809–819, 1988.

[Lipson *et al.*, 1997] P. Lipson, W.E. L. Grimson, and P. Sinha. Configuration based scene classification and image indexing. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1007–13, 1997.

[Liu and Picard, 1996] F. Liu and R.W. Picard. Detecting and segmenting periodic motion. Media lab vision and modelling tr-400, MIT, 1996.

[Longuet-Higgins, 1981] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[Loop, 1994] C. Loop. Smooth spline surfaces over irregular meshes. *Computer Graphics*, pages 303–310, 1994.

[Lorensen and Cline, 1987] W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21:163–169, 1987.

[Luong and Faugeras, 1996] Q.-T. Luong and O.D. Faugeras. The fundamental matrix: theory, algorithms, and stability analysis. *Int. J. of Comp. Vision*, 17(1):43–76, January 1996.

[Luong *et al.*, 1993] Q.-T. Luong, R. Deriche, O.D. Faugeras, and T. Papadopoulo. On determining the fundamental matrix: analysis of different methods and experimental results. Technical Report 1894, INRIA Sophia-Antipolis, 1993.

[Luong, 1992] Q.-T. Luong. *Matrice fondamentale et calibration visuelle sur l'environnement: vers une plus grande autonomie des systèmes robotiques.* PhD thesis, University of Paris XI, Orsay, France, 1992.

[Ma and Manjunath, 1997a] W. Y. Ma and B. S. Manjunath. Edgeflow: a framework for boundary detection and image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 744–749, 1997.

[Ma and Manjunath, 1997b] W.Y. Ma and B.S. Manjunath. Netra: a toolbox for navigating large image databases. In *IEEE Int. Conf. Image Processing*, pages 568–571, 1997.

[Ma and Manjunath, 1998] W.Y. Ma and B.S. Manjunath. A texture thesaurus for browsing large aerial photographs. *Journal of the American Society for Information Science (special issue on AI Techniques for Emerging Information Systems Applications)*, 49(7):633–648, 1998.

[Macaulay, 1916] F.S. Macaulay. *The Algebraic Theory of Modular Systems.* Cambridge University Press, 1916.

[Mahamud and Hebert, 2000] S. Mahamud and M. Hebert. Iterative projective reconstruction from multiple views. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, Hilton Head, SC, June 2000. In press.

[Malik and Perona, 1989] J. Malik and P. Perona. A computational model of texture segmentation. In *Proc Asilomar conf. on Signals, Systems and Computers*, pages 490–4, 1989.

[Malik and Perona, 1990] J. Malik and P. Perona. Preattentive texture discrimination with early visual mechanisms. *J. Opt. Soc. America*, 7A(5):923–932, 1990.

[Malik, 1987] J. Malik. Interpreting line drawings of curved objects. *Int. J. of Comp. Vision*, 1(1):73–103, 1987.

[Maloney and Wandell, 1986] L.T. Maloney and B.A. Wandell. A computational model of colour constancy. *J. Opt. Soc. America - A*, 1:29–33, 1986.

[Maloney, 1984] L.T. Maloney. *Computational approaches to Colour Constancy.* PhD thesis, Stanford University, 1984.

[Maloney, 1986] L.T. Maloney. Evaluation of linear models of surface spectral reflectance with small numbers of parameters. *J. Opt. Soc. America*, 3:1673–1683, 1986.

[Manjunath and Ma, 1996a] B.S. Manjunath and W.Y. Ma. Browsing large satellite and aerial photographs. In *IEEE Int. Conf. Image Processing*, 1996.

[Manjunath and Ma, 1996b] B.S. Manjunath and W.Y. Ma. Texture features for browsing and retrieval of image data. *IEEE T. Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[Manocha, 1992] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics.* PhD thesis, Computer Science Division, Univ. of California at Berkeley, 1992.

[Marimont and Wandell, 1992] D.H. Marimont and B.A. Wandell. Linear models of surface and illuminant spectra. *J. Opt. Soc. Am.-A*, 9:1905–1913, 1992.

[Marr and Nishihara, 1978] D. Marr and K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proc. Royal Society, London*, B-200:269–294, 1978.

[Marr and Poggio, 1976] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194:283–287, 1976.

[Marr and Poggio, 1979] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London*, B 204:301–328, 1979.

[Marr, 1977] D. Marr. Analysis of occluding contour. *Proc. Royal Society, London*, B-197:441–475, 1977.

[Marr, 1982] D. Marr. *Vision*. Freeman, San Francisco, 1982.

[Maybank and Faugeras, 1992] S.J. Maybank and O.D. Faugeras. A theory of self-calibration of a moving camera. *Int. J. of Comp. Vision*, 8(2):123–151, 1992.

[McCann *et al.*, 1976] J.J. McCann, S.P. McKee, and Taylor. Quantitative studies in retinex theory. *Vision Research*, 16:445–458, 1976.

[McKee *et al.*, 1990] S.P. McKee, D.M. Levi, and S.F. Brown. The imprecision of stereopsis. *Vision Research*, 30(11):1763–1779, 1990.

[McLachlan and Krishnan, 1996] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley and Sons, 1996.

[McMillan and Bishop, 1995] L. McMillan and G. Bishop. Plenoptic modeling: an image-based rendering approach. In *SIGGRAPH*, pages 39–46, Los Angeles, CA, August 1995.

[Milenkovic and Kanade, 1985] V.J. Milenkovic and T. Kanade. Trinocular vision using photometric and edge orientation constraints. In *Proc. DARPA Image Understanding Workshop*, pages 163–175, December 1985.

[Minka and Picard, 1997] T.P. Minka and R.W. Picard. Interactive learning with a "society of models". *Pattern Recognition*, 30:465–481, 1997.

[Minka, 1996] T. Minka. An image database browser that learns from user interaction. Mit media lab perceptual computing section tr 365, MIT, 1996.

[MMF, 1967] Photo James Baes MMF. Barbara Steele. *Midi/Minuit Fantastique*, (17):4–33, June 1967.

[Mohr *et al.*, 1992] R. Mohr, L. Morin, and E. Grosso. Relative positioning with uncalibrated cameras. In J. Mundy and A. Zisserman, editors, *Geometric Invariance in Computer Vision*, pages 440–460. MIT Press, Cambridge, Mass., 1992.

[Moravec, 1983] H.P. Moravec. The stanford cart and the cmu rover. *Proceedings of the IEEE*, 71(7), July 1983.

[Morgan, 1987] A.P. Morgan. *Solving Polynomial Systems using Continuation for Engineering and Scientific Problems*. Prentice Hall, Englewood Cliffs, NJ, 1987.

[Morita and Kanade, 1997] T. Morita and T. Kanade. A sequential factorization method for recovering shape and motion from image sequences. *IEEE Trans. Patt. Anal. Mach. Intell.*, 19(8), August 1997.

[Mundy and Vrobel, 1994] J.L. Mundy and P. Vrobel. The role of iu technology in radius phase ii. In *Proc. Image Understanding Workshop*, pages 251–64, 1994.

[Mundy, 1995] J.L. Mundy. The image understanding environment program. *IEEE Expert*, 10(6):64–73, 1995.

[Mundy, 1997] J.L. Mundy. Iu for military and intelligence applications, how automatic will it get? In *25'th AIPR workshop. Emerging applications of computer vision, Proc SPIE*, volume 2962, pages 162–170, 1997.

[Nalwa, 1988] V.S. Nalwa. Line-drawing interpretation: A mathematical framework. *Int. J. of Comp. Vision*, 2:103–124, 1988.

[Navy, 1969] US Navy. *Basic Optics and Optical Instruments*. Dover, 1969. Prepared by the Bureau of Naval Personnel.

[Nayar and Oren, 1995] S.K. Nayar and M. Oren. Visual appearance of matte surfaces. *Science*, 267(5201):1153–1156, 1995.

[Niem and Buschmann, 1994] W. Niem and R. Buschmann. Automatic modelling of 3D natural objects from multiple views. In *European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production*, Hamburg, Germany, 1994.

[Nitzan, 1988] D. Nitzan. Three-dimensional vision structure for robot applications. *IEEE Trans. Patt. Anal. Mach. Intell.*, 10(3):291–309, May 1988.

[Niyogi and Adelson, 1995] S.A. Niyogi and E.H. Adelson. Analyzing and recognizing walking figures in xyt. Media lab vision and modelling tr-223, MIT, 1995.

[Noble *et al.*, 1997] A. Noble, D. Wilson, and J. Ponce. On Computing Aspect Graphs of Smooth Shapes from Volumetric Data. *Computer Vision and Image Understanding: special issue on Mathematical Methods in Biomedical Image Analysis*, 66(2):179–192, 1997.

[Ogle and Stonebraker, 1995] V.E. Ogle and M. Stonebraker. Chabot: retrieval from a relational database of images. *Computer*, 28:40–8, 1995.

[Ohta and Kanade, 1985] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search. *IEEE Trans. Patt. Anal. Mach. Intell.*, 7(2):139–154, 1985.

[Ohta *et al.*, 1981] Y. Ohta, K. Maenobu, and T. Sakai. Obtaining surface orientation from texels under perspective projection. In *Proc. International Joint Conference on Artificial Intelligence*, pages 746–751, 1981.

[Oja, 1983] E. Oja. *Subspace methods of pattern recognition.* Research Study Press, 1983.

[Okutami and Kanade, 1993] M. Okutami and T. Kanade. A multiple-baseline stereo system. *IEEE Trans. Patt. Anal. Mach. Intell.*, 15(4):353=363, 1993.

[Okutami and Kanade, 1994] M. Okutami and T. Kanade. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Trans. Patt. Anal. Mach. Intell.*, 16(9), 1994.

[Oren and Nayar, 1995] M. Oren and S.K. Nayar. Generalization of the lambertian model and implications for machine vision. *Int. J. Computer Vision*, 14(14):227–251, 1995.

[Oren *et al.*, 1997] M. Oren, C. Papageorgiou, P. Sinha, and E. Osuna. Pedestrian detection using wavelet templates. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 193–9, 1997.

[Pae and Ponce, 1999] S. Pae and J. Ponce. Toward a scale-space aspect graph: Solids of revolution. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, volume II, pages 196–201, Fort Collins, CO, June 1999.

[Pentland *et al.*, 1996] A. Pentland, R. Picard, and S. Sclaroff. Photobook: content-based manipulation of image databases. *Int. J. Computer Vision*, 18(3):233–54, 1996.

[Pentland, 1986] A.P. Pentland. Perceptual organization and the representation of natural form. *Artificial Intelligence Journal*, 28:293–331, 1986.

[Peri and Nayar, 1997] V.N. Peri and S.K. Nayar. Generation of perspective and panoramic video from omnidirectional video. In *Proc. DARPA Image Understanding Workshop*, New Orleans, LA, May 1997.

[Perona and Malik, 1990] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Patt. Anal. Mach. Intell.*, 12(7):629–639, July 1990.

[Petitjean *et al.*, 1992] S. Petitjean, J. Ponce, and D.J. Kriegman. Computing exact aspect graphs of curved objects: Algebraic surfaces. *Int. J. of Comp. Vision*, 9(3):231–255, 1992.

[Petitjean, 1995] S. Petitjean. *Géométrie énumérative et contacts de variétés linéaires: application aux graphes d'aspects d'objets courbes.* PhD thesis, Institut National Polytechnique de Lorraine, 1995.

[Picard and Minka, 1995] R.W. Picard and T. Minka. Vision texture for annotation. *J. Multimedia systems*, 3:3–14, 1995.

[Plantinga and Dyer, 1990] H. Plantinga and C. Dyer. Visibility, occlusion, and the aspect graph. *Int. J. of Comp. Vision*, 5(2):137–160, 1990.

[Platonova, 1981] O.A. Platonova. Singularities of the mutual disposition of a surface and a line. *Russian Mathematical Surveys*, 36(1):248–249, 1981.

[Poelman and Kanade, 1997] C.J. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. *IEEE Trans. Patt. Anal. Mach. Intell.*, 19(3):206–218, March 1997.

[Poggio and Sung, 1995] T. Poggio and Kah-Kay Sung. Finding human faces with a gaussian mixture distribution-based face model. In *Asian Conf. on Computer Vision*, pages 435–440, 1995.

[Pollard *et al.*, 1970] S.B. Pollard, J.E.W Mayhew, and J.P Frisby. A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449–470, 1970.

[Pollefeys *et al.*, 1999] M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *Int. J. of Comp. Vision*, 32(1):7–26, August 1999.

[Pollefeys, 1999] M. Pollefeys. *Self-calibration and metric 3D reconstruction from uncalibrated image sequences.* PhD thesis, Katholieke Universiteit Leuven, 1999.

[Ponce and Brady, 1987] J. Ponce and J.M. Brady. Toward a surface primal sketch. In T. Kanade, editor, *Three-dimensional machine vision*, pages 195–240. Kluwer Publishers, 1987.

[Ponce *et al.*, 1993] J. Ponce, T.A. Cass, and D.H. Marimont. Relative stereo and motion reconstruction. Technical Report UIUC-BI-AI-RCV-93-07, Beckman Institute, University of Illinois, 1993.

[Ponce, 2000] J. Ponce. Metric upgrade of a projective reconstruction under the rectangular pixel assumption. Manuscript, 2000.

[Pritchett and Zisserman, 1998] P. Pritchett and A. Zisserman. Wide baseline stereo matching. In *Proc. Int. Conf. Comp. Vision*, pages 754–760, Bombay, India, 1998.

[Psarrou *et al.*, 1997] A. Psarrou, V. Konstantinou, P. Morse, and P. O'Reilly. Content based search in mediaeval manuscripts. In *TENCON-97 - Proc. IEEE Region 10 Conf. Speech and Image technologies for computing and telecommunications*, pages 187–190, 1997.

[Rieger, 1987] J.H. Rieger. On the classification of views of piecewise-smooth objects. *Image and Vision Computing*, 5:91–97, 1987.

[Rieger, 1990] J.H. Rieger. The geometry of view space of opaque objects bounded by smooth surfaces. *Artificial Intelligence Journal*, 44(1-2):1–40, July 1990.

[Rieger, 1992] J.H. Rieger. Global bifurcations sets and stable projections of nonsingular algebraic surfaces. *Int. J. of Comp. Vision*, 7(3):171–194, 1992.

[Ripley, 1996] B.D. Ripley. *Pattern recognition and neural networks.* Cambridge University Press, 1996.

[Robert and Faugeras, 1991] L. Robert and O.D. Faugeras. Curve-based stereo: figural continuity and curvature. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 57–62, Maui, Hawaii, 1991.

[Robert and Faugeras, 1995] L. Robert and O.D. Faugeras. What can two images tell us about a third one? *Int. J. of Comp. Vision*, 16(1):35–56, 1995.

[Rowley *et al.*, 1996a] H.A. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing 8*, pages 875–881, 1996.

[Rowley *et al.*, 1996b] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 203–8, 1996.

[Rowley *et al.*, 1998a] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE T. Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[Rowley *et al.*, 1998b] H.A. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 38–44, 1998.

[Rubner *et al.*, 1998] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Int. Conf. on Computer Vision*, pages 59–66, 1998.

[Saito *et al.*, 1999] H. Saito, S. Baba, M. Kimura, S. Vedula, and T. Kanade. Appearance-based virtual view generation of temporally-varying events from multicamera images in the 3D room. Technical Report CMU-CS-99-127, Carnegie-Mellon University, 1999.

[Samuel, 1988] P. Samuel. *Projective Geometry.* Springer-Verlag, 1988. English translation of "Géométrie Projective", Presses Universitaires de France, 1986.

[Sawhney and Ayer, 1996] H. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *IEEE T. Pattern Analysis and Machine Intelligence*, 18(8):814–30, 1996.

[Schmid and Mohr, 1997a] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Trans. Patt. Anal. Mach. Intell.*, 19(5):530–535, May 1997.

[Schmid and Mohr, 1997b] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, May 1997.

[Seales and Dyer, 1991] W.B. Seales and C.R. Dyer. Constrained viewpoint from occluding contour. In *IEEE Workshop on Directions in Automated "CAD-Based" Vision*, pages 54–63, Maui, Hawaii, June 1991.

[Seitz and Dyer, 1995] S.M. Seitz and C.R. Dyer. Physically-valid view synthesis by image interpolation. In *Workshop on Representations of Visual Scenes*, Boston, MA, 1995.

[Seitz and Dyer, 1996] S.M. Seitz and C.R. Dyer. Toward image-based scene representation using view morphing. Technical Report 1298, University of Wisconsin, Madison, WI, 1996.

[Seloff, 1990] G.A. Seloff. Automated access to the nasa-jsc image archives. *Library Trends*, 38(4):682–696, 1990.

[Shashua, 1993] A. Shashua. Projective depth: a geometric invariant for 3D reconstruction from two perspective/orthographic views and for visual recognition. In *Proc. Int. Conf. Comp. Vision*, pages 583–590, Berlin, Germany, 1993.

[Shashua, 1995] A. Shashua. Algebraic functions for recognition. *IEEE Trans. Patt. Anal. Mach. Intell.*, 17(8):779–789, August 1995.

[Shi and Malik, 97] J. Shi and J. Malik. Normalised cuts and image segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 731–737, 97.

[Shimshoni and Ponce, 1997] I. Shimshoni and J. Ponce. Finite-resolution aspect graphs of polyhedral objects. *IEEE Trans. Patt. Anal. Mach. Intell.*, 19(4):315–327, 1997.

[Shirai, 1972] Y. Shirai. Recognition of polyhedrons with a range finder. *Pattern Recognition*, 4:243–250, 1972.

[Shirman and Sequin, 1987] L. Shirman and C. Sequin. Local surface interpolation with Bezier patches. *CAGD*, 4:279–295, 1987.

[Shum and Szeliski, 1998] H.Y. Shum and R. Szeliski. Construction and refinement of panoramic mosaics with global and local alignment. In *Proc. Int. Conf. Comp. Vision*, pages 953–958, Bombay, India, 1998.

[Sidenbladh *et al.*, 2000] Hedvig Sidenbladh, Michael J. Black, and David J. Fleet. Stochastic tracking of 3d human figures using 2d image motion. In *European Conference on Computer Vision*, 2000.

[Sillion, 1994] F. Sillion. *Radiosity and global illumination.* Morgan-Kauffman, 1994.

[Simon *et al.*, 1994] D. Simon, M. Hebert, and T. Kanade. Real-time 3D pose estimation using a high-speed range sensor. In *IEEE Int. Conf. on Robotics and Automation*, pages 2235–2241, San Diego, CA, 1994.

[Slama *et al.*, 1980] C.C. Slama, C. Theurer, and S.W. Henriksen, editors. *Manual of photogrammetry.* American Society of Photogrammetry, 1980. Fourth edition.

[Smith and Chang, 1996] J.R. Smith and S-F. Chang. Visualseek: A fully automated content-based image query system. In *ACM Multimedia Conference*, 1996.

[Smith and Chang, 1997] J.R. Smith and S-F. Chang. Visually searching the web for content. *IEEE Multimedia*, 4(3):12–20, 1997.

[Smith and Christel, 1995] M.A. Smith and M.G. Christel. Automating the creation of a digital video library. In *ACM Multimedia*, 1995.

[Smith and Hauptmann, 1995] M.A. Smith and A. Hauptmann. Text, speech and vision for video segmentation: The informedia project. In *AAAI Fall 1995 Symposium on Computational Models for Integrating Language and Vision*, 1995.

[Smith and Kanade, 1997] M. Smith and T. Kanade. Video skimming for quick browsing based on audio and image characterization. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 1997.

[Smith, 1996] T.R. Smith. A digital library for geographically referenced materials. *Computer*, 29(5):54–60, 1996.

[Snapper and Troyer, 1989] E. Snapper and R.J. Troyer. *Metric Affine Geometry.* Dover Publications Inc., 1989. Reprinted from Academic Press, 1971.

[Spetsakis and Aloimonos, 1990] M.E. Spetsakis and Y. Aloimonos. Structure from motion using line correspondences. *Int. J. of Comp. Vision*, 4(3):171–183, 1990.

[Srivastava and Ahuja, 1990] S. Srivastava and N. Ahuja. Octree generation from object silhouettes in perspective views. *Computer Vision, Graphics and Image Processing*, 49(1):68–84, 1990.

[Stein and Medioni, 1992] F. Stein and G. Medioni. Structural indexing: efficient 3D object recognition. *IEEE Trans. Patt. Anal. Mach. Intell.*, 14(2), 1992.

[Stewman and Bowyer, 1987] J. Stewman and K.W. Bowyer. Aspect graphs for planar-face convex objects. In *Proc. IEEE Workshop on Computer Vision*, pages 123–130, Miami, FL, 1987.

[Stewman and Bowyer, 1988] J. Stewman and K.W. Bowyer. Creating the perspective projection aspect graph of polyhedral objects. In *Proc. Int. Conf. Comp. Vision*, pages 495–500, Tampa, FL, 1988.

[Struik, 1988] D.J. Struik. *Lectures on classical differential geometry.* Dover, 1988. Reprint of the second edition (1961) of the work first published by Addison-Wesley in 1950.

[Sturm and Triggs, 1996] P. Sturm and B. Triggs. A factorization-based algorithm for multi-image projective structure and motion. In *Proc. European Conf. Comp. Vision*, pages 709–720, 1996.

[Sugihara, 1984] K. Sugihara. An algebraic approach to the shape-from-image-problem. *Artificial Intelligence Journal*, 23:59–95, 1984.

[Sullivan and Ponce, 1998] S. Sullivan and J. Ponce. Automatic model construction, pose estimation, and object recognition from photographs using triangular splines. *IEEE Trans. Patt. Anal. Mach. Intell.*, 20(10):1091–1096, Oct. 1998.

[Sullivan *et al.*, 1994] S. Sullivan, L. Sandford, and J. Ponce. Using geometric distance fits for 3D object modelling and recognition. *IEEE Trans. Patt. Anal. Mach. Intell.*, 16(12):1183–1196, December 1994.

[Sung and Poggio, 1998] K-K. Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE T. Pattern Analysis and Machine Intelligence*, 20:39–51, 1998.

[Swain and Ballard, 1991] M.J. Swain and D.H. Ballard. Color indexing. *Int. J. Computer Vision*, 7(1):11–32, 1991.

[Taubin *et al.*, 1994] G. Taubin, F. Cukierman, S. Sullivan, J. Ponce, and D.J. Kriegman. Parameterized families of polynomials for bounded algebraic and surface curve fitting. *IEEE Trans. Patt. Anal. Mach. Intell.*, 16(3):287–303, March 1994.

[Terzopoulos, 1984] D. Terzopoulos. *Multiresolution Computation of Visible-Surface Representations.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[Thimbley *et al.*, 1994] H.W. Thimbley, S. Inglis, and I.H. Witten. Displaying 3D images: algorithms for single-image random dot stereograms. *IEEE Computer*, pages 38–48, 1994.

[Thom, 1972] R. Thom. *Structural Stability and Morphogenesis.* Benjamin, New-York, 1972.

[Thompson and Mundy, 1987] D.W. Thompson and J.L. Mundy. Three-dimensional model matching from an unconstrained viewpoint. In *IEEE Int. Conf. on Robotics and Automation*, pages 208–220, Raleigh, NC, April 1987.

[Thompson *et al.*, 1966] M.M. Thompson, R.C. Eller, W.A. Radlinski, and J.L. Speert, editors. *Manual of Photogrammetry.* American Society of Photogrammetry, 1966. Third Edition.

[Todd, 1946] J.A. Todd. *Projective and Analytical Geometry.* Pitman Publishing Corporation, New York – Chicago, 1946.

[Tomasi and Kanade, 1992] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. of Comp. Vision*, 9(2):137–154, 1992.

[Torr, 1997] P.H.S. Torr. An assessment of information criteria for motion based model selection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 47–52, 1997.

[Triggs, 1995] B. Triggs. Matching constraints and the joint image. In *Proc. Int. Conf. Comp. Vision*, pages 338–343, Boston, MA, 1995.

[Tsai and Huang, 1984] R.Y. Tsai and T.S. Huang. Uniqueness and estimation of 3D motion parameters of rigid bodies with curved surfaces. *IEEE Trans. Patt. Anal. Mach. Intell.*, 6:13–27, 1984.

[Tsai, 1987a] R.Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, 1987.

[Tsai, 1987b] R.Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras. *Journal of Robotics and Automation*, RA-3(4):323–344, 1987.

[Turk and Levoy, 1994] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *SIGGRAPH*, pages 311–318, Orlando, Fla, July 1994.

[Ullman and Basri, 1991] S. Ullman and R. Basri. Recognition by linear combination of models. *IEEE Trans. Patt. Anal. Mach. Intell.*, 13(10):992–1006, 1991.

[Ullman, 1979] S. Ullman. *The Interpretation of Visual Motion.* The MIT Press, Cambridge, MA, 1979.

[Ullman, 1996] S. Ullman. *High-level Vision: Object Recognition and Visual Cognition.* MIT Press, 1996.

[Vaillant and Faugeras, 1992] R. Vaillant and O.D. Faugeras. Using extremal boundaries for 3D object modeling. *IEEE Trans. Patt. Anal. Mach. Intell.*, 14(2):157–173, February 1992.

[Viéville and Faugeras, 1995] T. Viéville and O. Faugeras. Motion analysis with a camera with unknown, and possibly varying intrinsic parameters. In *Proc. Int. Conf. Comp. Vision*, pages 750–756, Boston, MA, 1995.

[vir,] Virage home page at `http://www.virage.com/`.

[Wactlar *et al.*, 1996] H. Wactlar, T. Kanade, M. Smith, and S. Stevens. Intelligent access to digital video: The informedia project. *IEEE Computer*, 29(5), 1996.

[Waltz, 1975] D. L. Waltz. Understanding line drawings of scenes with shadows. In P.H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, New York, 1975.

[Wang and Freeman, 1990] R. Wang and H. Freeman. Object recognition based on characteristic views. In *International Conference on Pattern Recognition*, pages 8–12, Atlantic City, NJ, June 1990.

[Watts, 1987] N. Watts. Calculating the principal views of a polyhedron. CS Tech. Report 234, Rochester University, 1987.

[Weinshall and Tomasi, 1995] D. Weinshall and C. Tomasi. Linear and incremental acquisition of invariant shape models from image sequences. *IEEE Trans. Patt. Anal. Mach. Intell.*, 17(5), May 1995.

[Weng *et al.*, 1992] J. Weng, T.S. Huang, and N. Ahuja. Motion and structure from line correspondences: closed-form solution, uniqueness, and optimization. *IEEE Trans. Patt. Anal. Mach. Intell.*, 14(3):318–336, March 1992.

[Wheatstone, 1838] C. Wheatstone. On some remarkable, and hitherto unobserved, phenomena of binocular vision. *Philosophical Transactions of the Royal Society. (London)*, 128:371–394, 1838.

[Whitney, 1955] H. Whitney. On singularities of mappings of Euclidean spaces. I. Mappings of the plane into the plane. *Annals of Mathematics*, 62(3):374–410, 1955.

[Whitted, 1980] T. Whitted. An improved illumination model for shaded display. *Comm. of the ACM*, 23(6):343–349, June 1980.

[Wilkinson and Reinsch, 1971] J.H. Wilkinson and C. Reinsch. *Linear Algebra - Vol. II of Handbook for Automatic Computation.* Springer-Verlag, New York, 1971. Chapter I.10 by G.H. Golub and C. Reinsch.

[Williams and Chen, 1993] L. Williams and E. Chen. View interpolation for image synthesis. *SIGGRAPH*, 1993.

[Wolff, 1994] L.B. Wolff. Diffuse-reflectance model for smooth dielectric surfaces. *Journal of the Optical Society of America A*, 11(11):2956–68, 1994.

[Wolff, 1996] L.B. Wolff. Generalizing lambert's law for smooth surfaces. In *European Conference on Computer Vision*, pages 40–53, 1996.

[Wong, 1998] S.T.C. Wong. Cbir in medicine: still a long way to go. In *IEEE Workshop on Content Based Access of Image and Video Libraries*, page 114, 1998.

[Wren *et al.*, 1995] C. Wren, A. Azabayejani, T. Darrell, and A. Pentland. Pfinder: real-time tracking of the human body. Mit media lab perceptual computing section tr 353, MIT, 1995.

[Yachida *et al.*, 1986] M. Yachida, Y. Kitamura, and M. Kimachi. Trinocular vision: new approach for correspondence problem. In *Proceedings IAPR International Conference on Pattern Recognition*, pages 1041–1044, 1986.

[Yacoob and Davis, 2000] Yaser Yacoob and Larry S. Davis. Learned models for estimation of rigid and articulated human motion from stationary or moving camera. *Int. J. Computer Vision*, 36:5–30, 2000.

[Ziv and Lempel, 1977] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23:337–343, 1977.